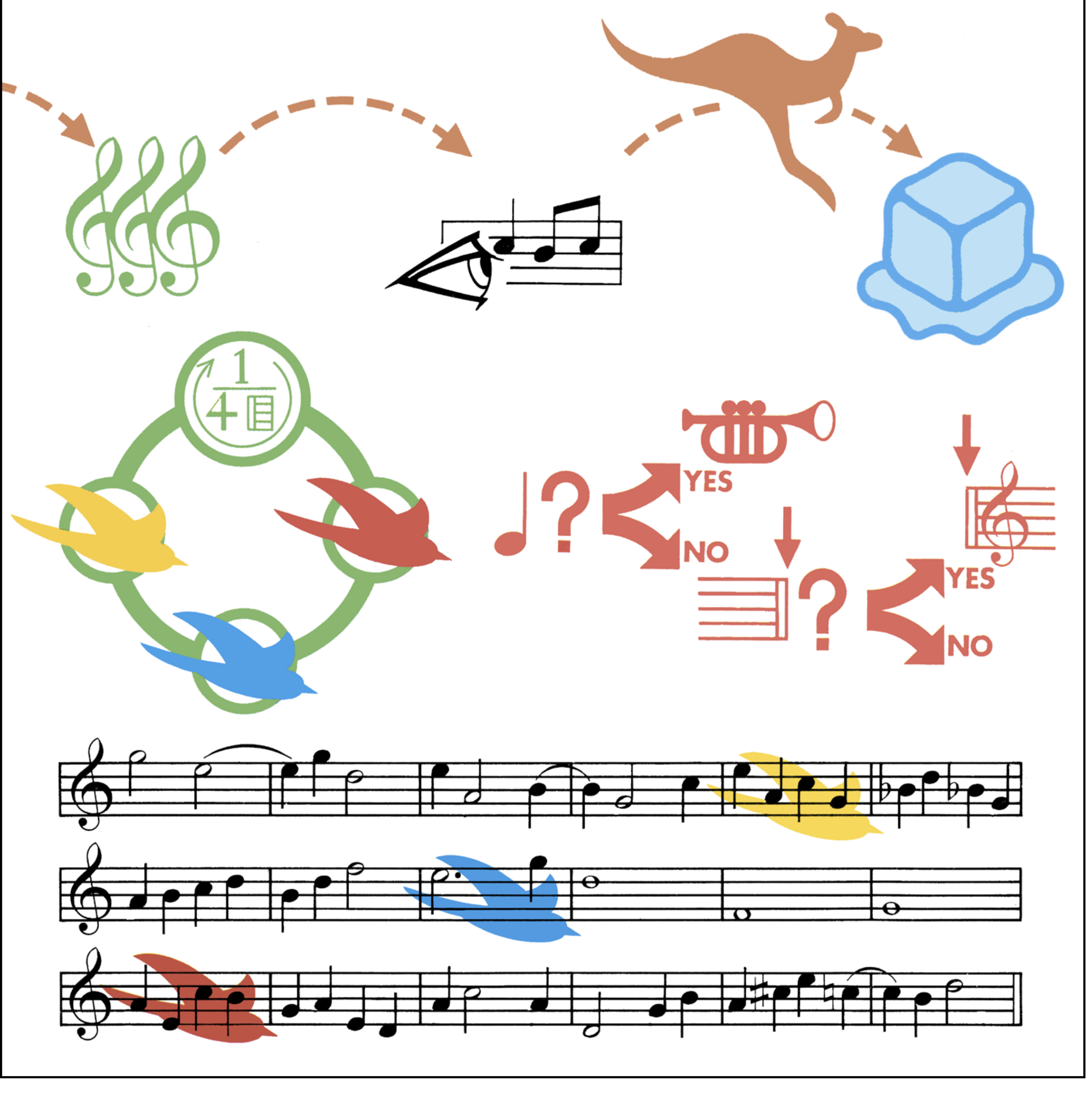
INVESTIGACION Y CIENCIA

Edición en español de **SCIENTIFIC AMERICAN**



LA PROGRAMACION DE LOS COMPUTADORES

Noviembre 1984

450 Ptas.

Los espacios en gris
corresponden a publicidad
en la edición impresa

Número 98
Noviembre 1984

INVESTIGACION Y
CIENCIA *Edición en español de* SCIENTIFIC AMERICAN

Colaboradores de este número:

Asesoramiento y traducción:
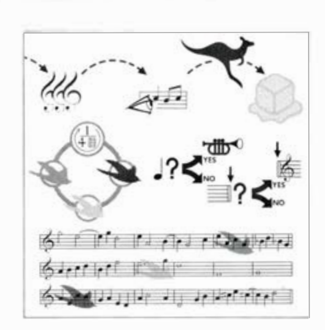
Luis Bou: *Programación de ordenadores, Algoritmos y estructuras de datos* y *Juegos de ordenador*; Jaime Marcos Escudero: *Lenguajes de programación*; Mateo Valero Cortés: *Sistemas operativos*; Departamento de Automática y Electrónica de la E.T.S. de Ingenieros Industriales de Zaragoza: *Programación y tratamiento de lenguajes*; José Oriol Costa Bou: *Programación de representaciones gráficas*; Ramón Pont Riba: *Programación del tratamiento de la información*; Juan P. Adrados Encinas: *Programación del control de procesos*; Monique Becue: *Programación en ciencias y matemáticas*; Jaime Agustí Cullell: *Programación de sistemas inteligentes*; J. Vilardell: *Taller y laboratorio*.

Ciencia y sociedad:

Pedro Laín Entralgo

Libros:

Diego Gracia y José Sanmartín



LA PORTADA

La ilustración de portada simboliza el tema de este número de INVESTIGACIÓN Y CIENCIA: el soporte lógico, o software, de ordenador. La propia ilustración es un programa, expresado en el lenguaje pictórico llamado Mandala, que ponen a punto Jaron Z. Lanier y colaboradores, de la VPL Research de Palo Alto, California. Para dar instrucciones al ordenador se sitúan en la pantalla de visualización una serie de pequeños símbolos gráficos, llamados iconos. Un canguro, en lo alto, brinca desde el icono que representa una triple clave de sol (activando así un programa para cánones tripartitos) hasta un icono que permite ver los datos musicales en la notación habitual, y después, a un cubito de hielo, donde se "congela" la secuencia de saltos. Un icono puede representar una jerarquía de estructuras de programación. La triple clave se "desarrolla" en un bucle –el aro– dibujado bajo ella. El bucle se ejecuta una sola vez, y va lanzando uno por uno los tres pájaros, a intervalos de cuatro compases, que ejecutan el canon.

Suscripciones:
Prensa Científica, S. A.
Calabria, 235-239
08029-Barcelona (España)
Teléfono 322 05 51 ext. 33-37

Condiciones de suscripción:
España:
Un año (12 números): 3850 pesetas
Extranjero:
Un año (12 números): 33 U.S. $
Ejemplar atrasado ordinario:
350 pesetas
Ejemplar atrasado extraordinario:
450 pesetas

Distribución para España:
Distribuciones de Enlace, S. A.
Ausias March, 49 - 08010-Barcelona

Distribución para los restantes países:
Editorial Labor, S. A.
Calabria, 235-239 - 08029-Barcelona

Publicidad:
Madrid:
Gustavo Martínez Ovin
Avda. de Moratalaz, 137 - 28030-Madrid
Teléfono 430 84 81
Cataluña:
Lourdes Padrós
Manuel Girona, 61, 2.º - 08034-Barcelona
Teléfono 204 45 83

Controlado por O.J.D.

PROCEDENCIA DE LAS ILUSTRACIONES

Pintura de la portada de Jerome Kuhl

Página	Fuente
14	Timothy C. May, Intel Corporation
16-18	Jerome Kuhl
20-22	Alan Kay
24-34	Alan D. Iselin
37	Steven P. Reiss, Universidad de Brown
38-45	Alan D. Iselin
48-54	Museo Cajal, Madrid
59-66	Gabor Kiss
71-82	Hank Iken, Walken Graphics
87	Ned Greene, Instituto de Tecnología de Nueva York
88	Ian Worpole
89	James K. Rinzler, Universidad de Brown
90	Ian Worpole
92	James K. Rinzler,
93	Evans & Sutherland
94	Lee Westover y Turner Whitted, Universidad de Carolina del Norte y Numerical Design Ltd.
95-96	Alvy Ray Smith, Lucasfilm Ltd.
97	Huseyin Kocak, Universidad de Brown
98	Bryce Flynn, The Picture Group, Inc.
100	Michael Lesk
102-107	Edward Bell
108	Michael Lesk
112	Aydin Controls
114-122	Hank Iken, Walken Graphics
125	Quesada/Burke
126-131	Ilil Arbel
132-134	Quesada/Burke
136	Ilil Arbel
141	Douglas B. Lenat, Universidad de Stanford
142-149	Ilil Arbel
151-154	Alan D. Iselin
156-160	Michael Goodman

ISSN 0210-136X
Dep. legal: B. 38.999-76
Fotocomposición Tecfa
Pedro IV, 160 - 08005-Barcelona
Fotocromos reproducidos por GINSA, S.A.
Imprime GRAFESA
Gráfica Elzeviriana, S.A.
Nápoles, 249 - Tel. 207 40 11
08013-Barcelona
Printed in Spain - Impreso en España

Los autores

ALAN KAY ("Programación de ordenadores") presta sus servicios en Apple Computer, Inc. Al tiempo que trataba de hacer carrera como músico de jazz, Kay se licenció en exactas y biología molecular por la Universidad de Colorado en Boulder. Se doctoró por la de Utah. Ingresó en el Laboratorio de Inteligencia Artificial de la Universidad de Stanford y fue uno de los miembros fundadores del Centro de Investigación de Palo Alto (PARC), creado por la Xerox Corporation en 1971. En él, Kay colaboró en la puesta a punto de un prototipo del primer ordenador personal. Kay tuvo participación directa en dos desarrollos típicos de los ordenadores personales: las "ventanas", zonas diferenciadas de la presentación en pantalla del ordenador, en las que pueden desarrollarse paralelamente tareas distintas, y el "ratón", un controlador de sobremesa que permite desplazar rápidamente un cursor electrónico en la pantalla. En 1981, Kay ingresó en Atari, Inc., donde dirigió el gabinete de investigación hasta el pasado mes de mayo.

NIKLAUS WIRTH ("Algoritmos y estructuras de datos") encabeza la división de informática del Instituto Politécnico Federal de Tecnología (ETH) con sede en Zurich. Su entusiasmo juvenil por los aeromodelos teledirigidos le llevó a estudiar electrónica. Se licenció en ingeniería eléctrica por el ETH (1959) y la Universidad Laval de Quebec (1960). Prosiguió estudios en los Estados Unidos, en la Universidad de California en Berkeley, por la que se recibió de doctor en 1963. Allí nació su interés por los lenguajes de programación; siendo profesor adjunto en el recién creado departamento de ciencias de cómputo de Stanford, tomó parte, de 1963 a 1967, en el desarrollo del lenguaje Algol W. De vuelta en Suiza, creó el lenguaje de programación Pascal, un lenguaje estructurado. Recientemente, Wirth ha explorado la confección del soporte físico (*hardware*) a la medida del lógico (*software*).

LAWRENCE G. TESLER ("Lenguajes de programación") dirige un grupo de desarrollo de programas de la División Macintosh de Apple Computer, Inc. Siendo alumno de ciencias de la Universidad de Stanford fundó, en 1965, una pequeña compañía de programación. Tras cinco años de experiencia en el campo comercial se incorporó al Laboratorio de Inteligencia Artificial de Stanford. En él investigó sobre simulación del conocimiento y preparación de formatos de documentos. En 1973 pasó al centro de investigación de la compañía Xerox en Palo Alto, donde su trabajo se centró en la programación de ordenadores personales. En 1980 ingresó en Appel para acometer el desarrollo de posibles aplicaciones del ordenador Lisa.

PETER J. DENNING y ROBERT L. BROWN ("Sistemas operativos") son especialistas en arquitectura de ordenadores del RIACS (Research Institute for Advanced Computer Science), que forma parte del Centro de Investigación Ames de la Administración Nacional para la Aeronáutica y el Espacio (NASA). Denning es el director del RIACS y Brown está adscrito a su plantilla. El primero se licenció en ingeniería eléctrica (1964) por el Manhattan College y se doctoró en esa misma especialidad por el Instituto de Tecnología de Massachusetts. Ha dictado cursos de esa materia en la Universidad de Princeton y de informática en la de Purdue. Brown estudió exactas en la Universidad Wesleyan de Ohio; prepara su doctorado en informática por la Universidad de Purdue.

TERRY WINOGRAD ("Programación y tratamiento de lenguajes") enseña informática y lingüística en Stanford. Inició su formación en el Colorado College, prosiguiéndola en el Instituto de Tecnología de Massachusetts (MIT); se recibió de doctor en matemáticas aplicadas en 1970 por este último centro. Enseñó en el MIT hasta 1973; ingresó entonces en el claustro docente de Stanford. Desde ese año trabaja como asesor del Centro de Investigación de Palo Alto de la Xerox Corporation. Winograd investiga sobre inteligencia artificial, lingüística informática y modelos de conocimiento.

ANDRIES VAN DAM ("Programación de representaciones gráficas") es jefe del departamento de informática de la Universidad de Brown. De origen holandés, estudió en el Swarthmore College y la Universidad de Pennsylvania, por la que se doctoró en ciencias de la computación en 1966 (el segundo doctorado de esa especialidad concedido en los Estados Unidos). Intervino en la fundación del departamento de informática de la Universidad de Brown y ha participado directamente en la instalación de terminales de trabajo en todas sus dependencias. Van Dam es coautor de *Fundamentals of Interactive Computer Graphics*.

MICHAEL LESK ("Programación del tratamiento de la información") dirige la división de investigación informática de la Bell Communications Research Inc. en Murray Hill, empresa en la que ingresó al poco de doctorarse en química física por la Universidad de Harvard (1969). Su interés por las bases de datos le ha llevado a desarrollar un programa de búsqueda de rutas para automóviles, así como a experimentar un sistema computarizado de catalogación para bibliotecas.

ALFRED Z. SPECTOR ("Programación del control de procesos") enseña informática en la Universidad de Carnegie-Mellon. Estudió en Harvard, licenciándose en matemáticas aplicadas; en la Universidad de Stanford se doctoró en ciencia de ordenadores en 1981. Mientras preparaba el doctorado trabajó en el laboratorio de investigación de IBM en San José. Tomó posesión de su cargo actual en 1981. Durante los dos últimos años ha colaborado en el diseño de un sistema integrado de ficheros para un proyecto de mecanización del campus, emprendido conjuntamente por Carnegie-Mellon e IBM.

STEPHEN WOLFRAM ("Programación en ciencias y matemáticas") pertenece al Instituto de Estudios Avanzados de Princeton desde 1982. Nacido en Londres, se educó en Eton y en la Universidad de Oxford. Se trasladó luego a los Estados Unidos, doctorándose en 1979, en física teórica, por el Instituto de Tecnología de California (Cal Tech). En 1980 se incorporó al claustro docente del Cal Tech, donde permaneció hasta ocupar su cargo actual. Wolfram ha trabajado en física de altas energías, cosmología y mecánica estadística.

DOUGLAS B. LENAT ("Programación de sistemas inteligentes") es especialista en inteligencia artificial. Enseña informática en la Universidad de Stanford. Licenciado por la de Pennsylvania, en 1976 se doctoró en informática por la Universidad de Stanford. En su tesis doctoral demostró que podían programarse los ordenadores para que encontraran teoremas matemáticos originales. Desde entonces sigue investigando la naturaleza del razonamiento heurístico. Durante un año enseñó en la Universidad Carnegie-Mellon.

Hace...

José M.ª López Piñero

... trescientos años

Nació en Madrid Martín Martínez, uno de los protagonistas de la segunda fase de la renovación científica de la medicina española, durante el primer tercio del siglo XVIII. Cincuenta años después de su nacimiento (el 10 de noviembre de 1684), casi exactamente, murió en la misma ciudad (el 9 de octubre de 1734). Esta doble efemérides no ha motivado, que yo sepa, recuerdo alguno, como es habitual en las de casi todos los científicos españoles, mientras se multiplican hasta la náusea las conmemoraciones en torno a las cuatro figuras tópicas a las que reduce nuestra tradición científica una mezquina ignorancia cada vez menos perdonable.

Martín Martínez estudió medicina en la Universidad de Alcalá, en un momento en el que la gran fundación renacentista se había convertido en un reducto del tradicionalismo científico intolerante y cerrado a las novedades. Como principal libro de texto, utilizó la obra que treinta años antes había publicado Francisco Henríquez de Villacorta, catedrático de prima de medicina en dicha Universidad y cabeza de la reacción más intransigente frente al movimiento *novator* a finales del siglo XVII. Por su estricto atenimiento al galenismo tradicional y su capacidad para las sutilezas escolásticas, recibió el calificativo ambivalente de "Galeno español". Martínez lo juzgó duramente como "un ingenio nacido para corromper el entendimiento de la juventud".

El mismo año en el que terminó sus estudios en Alcalá (1706), Martínez ganó una plaza de médico en el Hospital General, de Madrid. En esta institución de importancia ascendente y en el ambiente científico de la corte recibió una formación que no había podido adquirir en las aulas complutenses y asimiló una mentalidad acorde con las corrientes médicas e intelectuales vigentes en la Europa de la época. Le influyeron directamente dos médicos extranjeros venidos a España junto a Felipe V: Florencio Kelli, a quien el monarca nombró "director regio" en el Hospital General madrileño y que fue su maestro como anatomista, y José Cervi, médico primario de Felipe V y principal impulsor de la institucionalización de la nueva medicina en la España de estos años. Bajo la protección de Cervi, Martínez tuvo una brillante carrera profesional, siendo nombrado "profesor público" de anatomía en el Hospital General, médico de cámara de la familia real y examinador del Protomedicato.

A finales del siglo XVII, los responsables del movimiento *novator*, que había roto abiertamente en España con el galenismo tradicional, habían sido seguidores del sistema iatroquímico. En la época de Martínez continuó habiendo en España algunos partidarios tardíos de la quemiatría y también de otros sistemas, entre los que destaca el iatromecánico. Sin embargo, la tendencia central de la renovación pasó a ser una vigorosa corriente antisistemática cuyo punto de partida fue un eclecticismo apoyado en un vago empirismo que invocaba, una vez más, la tradición hipocrática. La primera expresión madura de dicha corriente fue la llamada "medicina escéptica" y tuvo precisamente como máximo doctrinario a Martín Martínez. La postura defendida por éste continuaba basada en el eclecticismo frente al atenimiento a sistemas cerrados, pero lo asociaba al "empirismo racional" como opción metodológica concreta para integrar los datos procedentes de la observación clínica y los de las ciencias básicas, especialmente la anatomía y la fisiología experimental. Aparte de recurrir a la tradición hipocrática, Martínez se apoyó expresamente en la línea encabezada por Sydenham y Bacon. Las principales obras en las que expuso este ideario fueron las tituladas *Philosophia sceptica* (1730) y *Medicina sceptica* (1722-25). Ambas están estructuradas en forma de diálogo, en los que un "escéptico o hipocrático" defiende los puntos de vista del autor frente a las "telarañas metafísicas" de un "aristotélico", un "cartesiano" y un "gassendista" y a la "inutilidad de las cuestiones médicas" de un "galénico" y un "(iatro)químico".

Como es lógico, estos dos libros de Martínez motivaron numerosas críticas por parte de los seguidores de las doctrinas tradicionales y también de los partidarios del sistema iatroquímico. De las publicadas por los galenistas, la más importante fue la del médico de cámara regio Bernardo López de Araujo (1725), a la que contestó Martínez en el segundo volumen de su *Medicina sceptica*. Araujo atacó también a Feijóo, siendo esta polémica el motivo inmediato de la profunda relación amistosa e intelectual que unió al médico madrileño y al benedictino gallego; Feijóo escribió la primera biografía de su amigo y la publicó en el volumen primero de su célebre *Theatro crítico universal*. Dejando aparte críticas de interés secundario, anotaremos también que la "medicina escéptica" de Martínez fue combatida por el iatroquímico Juan Gil Sanz en una obra, titulada expresivamente *Triunfo del ácido y álkali* (1728), que condujo a una nueva polémica.

La anatomía fue objeto de especial atención por parte de Martín Martínez. Tras un resumen en forma de diálogo destinado a los cirujanos romancistas (*Noches anatómicas o anatomía compendiosa*, 1716), publicó un tratado en el que recogió el contenido de sus lecciones en el Hospital General: *Anatomía completa del hombre, con todos los hallazgos, nuevas doctrinas y observaciones raras hasta el tiempo presente* (1728). Reeditado en siete ocasiones, la última de ellas en 1788, este tratado fue al mismo tiempo el texto anatómico más importante publicado en la España de la primera mitad del siglo XVIII y una de las obras fundamentales de su autor. Su prólogo incluye una dura denuncia del estado de la enseñanza médica en España, comparable por su rigor a las formuladas cuarenta años antes por Cabriada y otros *novatores*: "Con la ocasión de ser examinador del Protomedicato y pedir razón a algunos de la economía animal y de las metástases y otros fenómenos morbosos, no he podido oír sin pudor que los que pretenden ser médicos responden que de eso no saben, porque no han leído ni visto anatomía, ni se enseña en su universidad. Y si alguno se esfuerza a dar alguna noticia, apenas pasa de saber que el hígado está al lado derecho y el bazo al izquierdo. En ellos es disculpable, porque no se les puede pedir más cuenta que de los talentos que se les entregaron; el defecto está en la educación porque (ya se ve) los débiles maestros no pueden criar robustos discípulos. Otros más aplicados suelen adquirir algunas noticias con el tiempo y el trato, pero no sería cosa prodigiosa que un arquitecto jamás hubiese visto demostración alguna geométrica y un médico, después de cuarenta años de práctica, se vaya a la otra vida, sin haber visto una disección anatómica... Por este descuido que hay en nuestras escuelas de enseñar la anatomía y quimia (partes tan precisas para hacer un perfecto médico) nos critican los extranjeros...

"Conténtanse nuestras universidades con disputar puntos de menor impor-

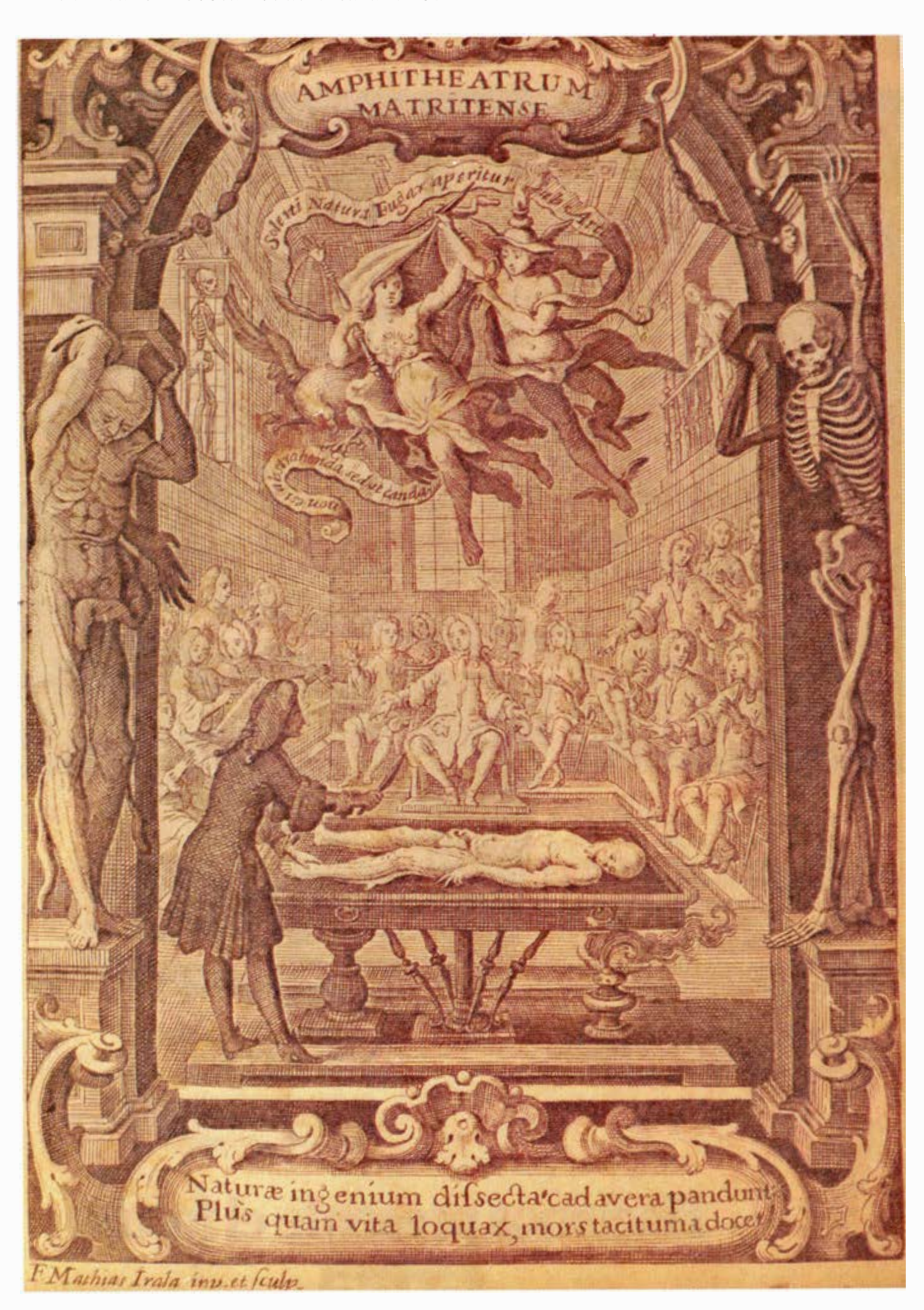
tancia y así, en lugar de una medicina útil, experimental y masculina, aprendemos una medicina femenil y contenciosa. Disputando el por qué se nos olvidó el cómo y abandonando el entendimiento el firme camino de la observación, se perdió en el laberinto de la conjetura. Murió Hipócrates y con él murió la medicina verdadera, porque faltó la aplicación observativa. Acá la anatomía se cree por fe, la probabilidad se trata como dogma y los fenómenos se interpretan a gusto, sin reparar que la experiencia suele burlar nuestra razón, pero la razón nunca desampara la experiencia.

"Con saber recetar cuatro tarazones de ruibarbo, una angélica o unos polvos escalfados y acotar un salpicón de textos (lo que puede hacer un curandero), hay quien se juzga más que Apolo, y es que conoce que esta es mercancía de despacho y que, aunque se empalidezca y críe flatos sobre los libros, ni por eso ha de subir más, ni ganar más."

Resulta muy significativo que, al criticar el criterio escolástico de autoridad, cite en este prólogo a Galileo: "No pocos hay tan asidos a lo que sus maestros aprendieron que, en oyendo algo nuevo, tocan a rebato y no hay forma de apartarlos de su opinión, aunque vean por sus mismos ojos lo contrario. De uno de estos refiere Galilei un cuento bien gracioso: estaba cierto célebre anatómico demostrando que el origen de los nervios era el cerebro y no el corazón, como quiso Aristóteles; hallóse presente un peripatético el cual, habiendo claramente visto que todos los nervios salían de un tronco medular que nacía del cerebro y que al corazón sólo entraban algunos pequeños ramillos, dijo: tan patente habeis puesto en los ojos al nacimiento de los nervios, que si el texto de Aristóteles no dijera lo contrario, casi estuviera por creerlo".

El contenio de la *Anatomía completa* no está basado en la experiencia disectiva de Martínez, aunque ocasionalmente recurra a ella y a las demostraciones que realizó junto a Florencio Kelli en el anfiteatro anatómico del Hospital General madrileño, a algunas de las cuales asistió el propio Felipe V. El texto depende en gran parte de *L'anatomie de l'homme*, del gran cirujano francés Pierre Dionis, resumen de sus cursos anatómicos en el Jardin-du-Roi, de París, publicado por vez primera en 1690 y reeditado después muchas veces y traducido a varios idiomas, entre ellos el chino. Por su orientación ideológica, rotundamente moderna, y por su procedencia de una institución científica innovadora, enfrentada con el anquilosamiento de la enseñanza universitaria, es lógico que el libro de Dionis se convirtiera en un modelo para el tratado de Martínez. No está justificado, sin embargo, el agresivo juicio del jesuita Hervás y Panduro (1800) que lo presentó casi como un plagio. En el terreno terminológico, Martínez intentó reaccionar contra la invasión del léxico anatómico castellano por latinismos y galicismos, defendiendo el uso de vocablos vulgares o procedentes de los grandes anatomistas españoles del siglo XVI. Fue el último paladín del casticismo en este terreno, ya que la influencia francesa acabaría por imponerse en el lenguaje anatómico castellano a partir de la segunda mitad del siglo XVIII. La obra incluye veinticinco láminas calcográficas, casi todas obra de Matías de Irala Yuso; solamente hay dos firmadas por otros grabadores, una de las cuales es un retrato de José Cervi, a quien va dedicado el libro. No ha sido estudiada la procedencia de estas láminas, que en su mayor parte son réplicas de ilustraciones anatómicas francesas y holandesas, aunque hay también grabados originales, alguno tan notable como el de Irala que representa el anfiteatro anatómico del Hospital General, de Madrid.

Este libro de Martínez no es solamente un tratado de anatomía descriptiva. Incluye asimismo un resumen de



1. Lámina de la **Anatomía completa del hombre** *(1728), de Martín Martínez, grabada por Matías Irala, que representa el anfiteatro anatómico del Hospital General, de Madrid. El propio Felipe V asistió en algunas ocasiones a las demostraciones anatómicas y experimentos fisiológicos que en él realizaban Martín Martínez y su maestro Florencio Kelli.*



2. Lámina, también grabada por Matías Irala, que ilustra el trabajo de Martín Martínez titulado **Observatio rara de corde in monstroso infantulo** *(1723). Incluye el estudio clínico y la autopsia de un caso de "cor protrussum" y fue la aportación de Martínez más difundida y estimada en la Europa de la época.*

morfología textural de acuerdo con la teoría fibrilar, numerosas noticias anatomopatológicas y, sobre todo, un estudio sistemático de la fisiología del cuerpo humano, que ocupa casi la mitad de sus páginas y se detiene especialmente en el estudio de la circulación, la respiración y las funciones del sistema nervioso central. Se trata de una de las primeras exposiciones didácticas de conjunto que se dedicaron a esta materia desde una mentalidad "moderna".

Entre las demás publicaciones de Martín Martínez sobresale la titulada *Observatio rara de corde in monstroso infantulo* (1723), en la que expuso el estudio clínico y la autopsia de un caso de "cor protrussum". Su rigurosidad refleja el progreso alcanzado por la investigación de las alteraciones teratológicas, que tradicionalmente habían sido mero objeto de la curiosidad más superficial: "En Madrid, reinando el muy poderoso, justo y gran monarca Felipe V, el 13 de enero de 1706, nació un niño que tenía en medio del tórax una tumoración carnosa con un movimiento continuo de carácter pulsátil.

"Invitado por un cirujano asistente a ver esta rareza, lo hice con gusto. Me encontré con un niño grande y robusto, de nueve meses, y, al desnudarlo, apareció en su pecho una tumoración carnosa. Se trataba de una masa musculosa y bastante compacta, de casi cuatro dedos de longitud y tres de anchura; tenía forma cónica y estaba situada horizontalmente; la punta, revestida de muy escasa adiposidad, miraba hacia adelante, mientras que la base estaba adherida a la parte central del tórax. Dos pequeñas prominencias salían de la base y descubrí un tubo blanquecino que partía de la izquierda. Tenía diástoles y sístoles alternativos tan vigorosos que, al apoyar la mano, la sacudía con fuerza...

"El niñito encontró una plácida muerte poco después de recibir las aguas bautismales...

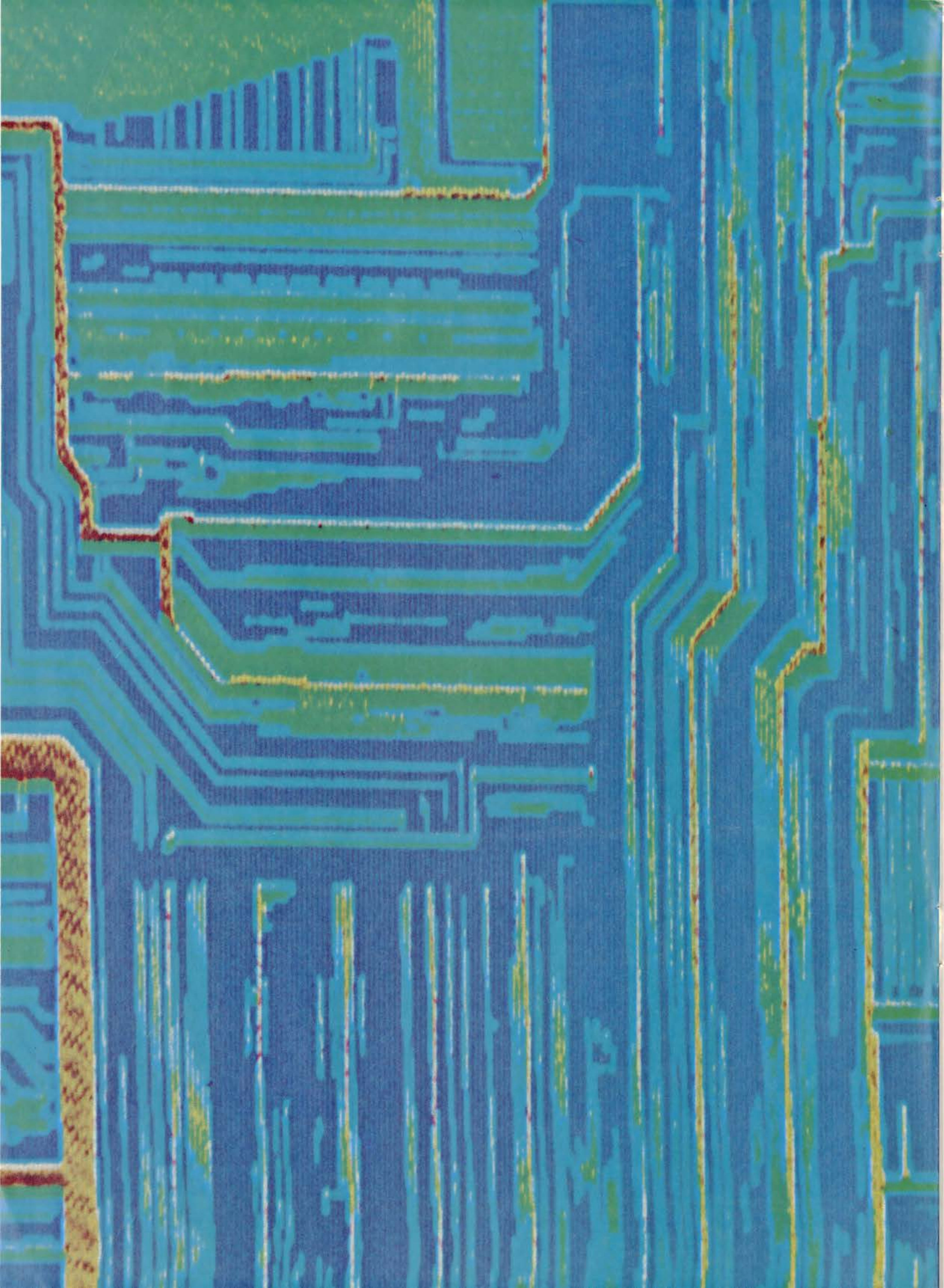
"*Disección anatómica del pequeño cadáver*... Al apoyar el escalpelo en la horquilla hasta llegar a la tumoración y cortar luego por debajo de ésta, apareció ante nuestros ojos un esternón hendido y fragmentado, desde las clavículas hasta su punta cartilaginosa. Una línea rojiza, como un rasguño o surco, correspondía por la parte de fuera a la fisura esternal.

"En la tumoración, encontramos una pequeña masa, envuelta en una túnica estrechamente adherida y carente de pericardio. Cuando la seccionamos por el lado izquierdo, apareció una cavidad cuyas paredes, llenas de surcos irregulares y de fosas, indicaban claramente que era el ventrículo izquierdo. Seccioné longitudinalmente con cuidado el vaso grande procedente del lado izquierdo del tórax y, en la parte cercana al ventrículo, aparecieron tres válvulas semilunares, demostrativas de que se trataba de la aorta, aunque su disposición no había sido antes descrita por los anatomistas.

"Vi también las aurículas de ambos lados, de las cuales la izquierda aparece claramente en la figura. Por no cansar al lector, diré solamente que encontramos los otros tres grandes vasos comunicados con el corazón, así como el ventrículo derecho y el tabique interventricular...

"Los pulmones estaban normalmente conformados y situados, pero quedaba vacío el espacio intermedio. Faltaba el pericardio y todos los grandes vasos se comunicaban con el corazón, a través de intersticios del esternón, en la forma fijada por la naturaleza. En consecuencia, era lícito concluir que la enfermedad no había consistido en la duplicación del corazón, ni tampoco en la alteración de su estructura, que se ajustaba a las leyes naturales, sino a un cambio de localización."

Este trabajo fue conocido y estimado en Europa. Albrecht Haller lo reeditó en el volumen segundo (1747) de sus *Disputationes anatomicae selectae*.



Programación de ordenadores

Presentación de un número monográfico sobre la teoría y la técnica que nos permiten que el ordenador cumpla nuestros designios. Son los programas lo que da forma y propósito a esas máquinas, como a la arcilla el escultor

Alan Kay

Los ordenadores son a la informática lo que los instrumentos a la música. La programación, o *software*, son las partituras, cuya interpretación potencia nuestros sentidos y facultades y eleva el espíritu. Dijo Leonardo da Vinci que la música era "la modelación del silencio"; aún más oportuna es su frase como descripción de los programas de ordenador. Nada tiene de misteriosa la invisible naturaleza de la música; tampoco la de los programas, o logicales, es más misteriosa que la desaparición del regazo al ponerse en pie. El verdadero misterio, que se explora en este número de INVESTIGACIÓN Y CIENCIA, es cómo lograr tantísimo con materiales tan simples, dada la arquitectura adecuada.

La materia prima de la informática son marcas o señales, de concisión extrema, almacenadas por miles de millones en la maquinaria de cómputo, el soporte físico, o *hardware*. En una partitura musical, los elementos materiales donde queda plasmada la melodía son tinta y papel pautado. En biología, el mensaje que de una a otra generación transmite el ADN se concreta y conserva en la organización y disposición de los grupos moleculares llamados nucleótidos. Lo mismo que han existido muchos materiales (desde la arcilla al papiro, pasando por el pergamino, hasta la tinta y el papel) donde almacenar y conservar los signos de la escritura, también en la técnica informática se ha recurrido a diversos sistemas materiales para almacenar sus signos: ejes giratorios, perforaciones en tarjetas, flujos magnéticos, válvulas electrónicas, transistores y circuitos integrados inscritos en diminutas pastillas de silicio. La capacidad de representación de los signos trazados en arcilla o papel, en el ADN, o en la memoria de los ordenadores, es en principio la misma, pues el único significado intrínseco de una marca o signo es encontrarse donde está. Afirma Gregory Bateson que "información es cualquier diferencia que crea una diferencia". La primera diferencia la establece el signo o marca; la segunda alude a la necesidad de interpretación.

La notación con que se especifican las musiquillas de ascensor es la misma que la de las fugas para órgano de Bach. En los ordenadores, una misma notación puede tanto designar tablas actuariales como dar vida a un nuevo mundo. De antiguo es sabido que la notación con que se escriben sonetos y pintadas es la misma. Que otro tanto sea cierto para los ordenadores puede contribuir a disipar gran parte del misterio de las nuevas tecnologías y asentar sobre más firmes bases el pensamiento, en lo que a ellas concierne.

Al igual que ocurre con casi todos los demás materiales de construcción (se trate de una catedral, una bacteria, un soneto, una fuga coral o un procesador de textos), la arquitectura domina la materia. Comprender el barro no es comprender la vasija. Lo que pueda ser una vasija se apreciará mejor conociendo a sus usuarios y creadores, y sus necesidades, tanto al informar con significado al material como al extraer significado de la forma.

En cuanto medio de expresión, sí hay diferencia cualitativa entre los ordenadores y la arcilla o el papel. Lo mismo que el aparato genético de una célula, el ordenador lee, escribe y saca consecuencias de sus propias marcas, hasta niveles de autointerpretación cuyos límites intelectuales no conocemos aún. Por consiguiente, la tarea de quien quiera comprender los programas no consiste, meramente, en percibir la vasija en lugar del barro, sino en vislumbrar en los trabajos de los principiantes (pues todos somos principiantes en la profesión de las ciencias de cómputo, que aún están echando pluma) la posibilidad del advenimiento de las porcelanas chinas o de Limoges.

No dedicaré aquí más tiempo a los métodos que en computación se utilizan para conservar y leer señales del que en biología molecular pueda dedicarse a estudiar las propiedades generales de los átomos. Una gran capacidad de almacenamiento de marcas, más un sencillísimo juego de instrucciones, son suficientes para construir cualesquiera mecanismos de representación necesarios, incluida la simulación de un nuevo ordenador completo. Augusta Ada, Condesa de Lovelace, la primera de los genios de la informática, que programó el "ingenio analítico" que Charles Babbage había diseñado, tuvo clara conciencia de la capacidad de simulación de una máquina universal. En el decenio de 1930, Alan M. Turing formulaba más nítidamente esta tesis, al probar que un mecanismo de notoria sencillez podía simular cualquier mecanismo.

La noción de que un ordenador cualquiera pueda simular a otro cualquiera, tanto existente como futuro, puede tal vez tener importancia filosófica, pero

1. UN MENSAJE INTANGIBLE que toma cuerpo en un medio material: tal es la esencia del soporte lógico de ordenador. El mensaje se ha hecho aquí visible trasladando a imagen contrastes de tensiones eléctricas, en una fotografía, tomada al microscopio electrónico de barrido, de una minúscula porción de un microprocesador Intel 80186. Los rasgos que la imagen muestra no son los transistores ni los conductores de la pastilla de microcircuitos, sino las señales que los atraviesan. Las trayectorias de los electrones secundarios emitidos en respuesta al haz del microscopio son perturbadas por los campos electromagnéticos de la superficie de la pastilla. Las regiones de potenciales altos atraen electrones, debilitando la señal formadora de imagen. El haz del microscopio se proyecta solamente cuando el microprocesador se encuentra en un determinado estado electrónico, es decir, cuando ciertos elementos lógicos están en estado excitado. Los colores de las líneas indican las tensiones eléctricas de las líneas de comunicación, metálicas, que llevan a elementos lógicos. Donde haya una señal viajando a lo largo de una línea habrá una región de voltaje alto. La imagen (en falso color) ha sido procesada de modo que tales regiones, y por tanto los "mensajes" que transmiten, se vean de azul claro. Las regiones de tensión baja son verdes y, las de tensiones medias, amarillas. Las líneas rojas indican conductores a potencial de tierra, esto es, cero volt.

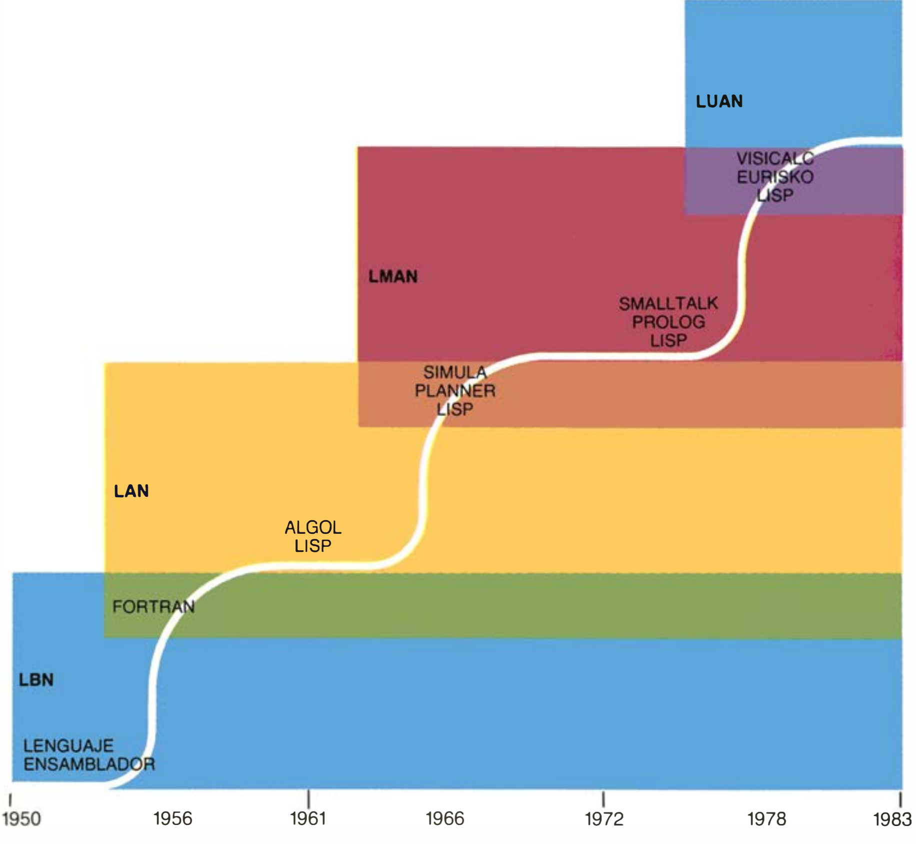
no es la solución de todos los problemas de computación. Con demasiada frecuencia, ordenadores sencillos que pretenden pasar por mucho más perfectos quedan atascados en el "atolladero de Turing"; para nada nos sirven si el resultado ha de estar disponible en menos de un millón de años. Con otras palabras, también las mejoras cuantitativas pueden ser útiles. Pensemos como, al acelerar el paso de una película desde dos hasta 20 fotogramas por segundo (un mero orden de magnitud), se produce una notable diferencia: la percepción subjetiva de continuidad de movimiento. Gran parte de la "vida" de la interacción visual y auditiva depende de su ritmo.

Descubrimos de niños que la arcilla puede moldearse, que para darle forma basta hundir las manos en su masa. Pocos hemos podido aprender tal cosa en los ordenadores. Su material se nos presenta tan desligado de la experiencia humana como un lingote radiactivo, que hay que manejar desde lejos, mediante botones, brazos mecánicos y un monitor de televisión. ¿Qué clase de contacto emocional podemos establecer con este nuevo tipo de material plástico, si tan remoto se nos presenta su acceso físico?

Podemos sentir lo que llamaríamos "barro computacional" gracias a la "interfase de usuario", es decir, la programación que media entre las personas y los programas específicos que conforman al ordenador en instrumento para una finalidad concreta, tanto si ésta consiste en proyectar un puente como si se trata de escribir un artículo. Hubo tiempos en que la interfase con el usuario era lo último que se diseñaba de un sistema. Hoy es lo primero. Se reconoce su fundamental y primario carácter, pues tanto a noveles como a profesionales lo que se presenta ante nuestros sentidos *es* nuestro ordenador. La "ilusión de usuario", como la llamamos mis colegas y yo en el Centro de Investigación Xerox de Palo Alto, es una justificación mítica y simplista que cada cual se construye para explicar (y conjeturar) las acciones del sistema, y para decidir qué se debe hacer a continuación en cada caso.

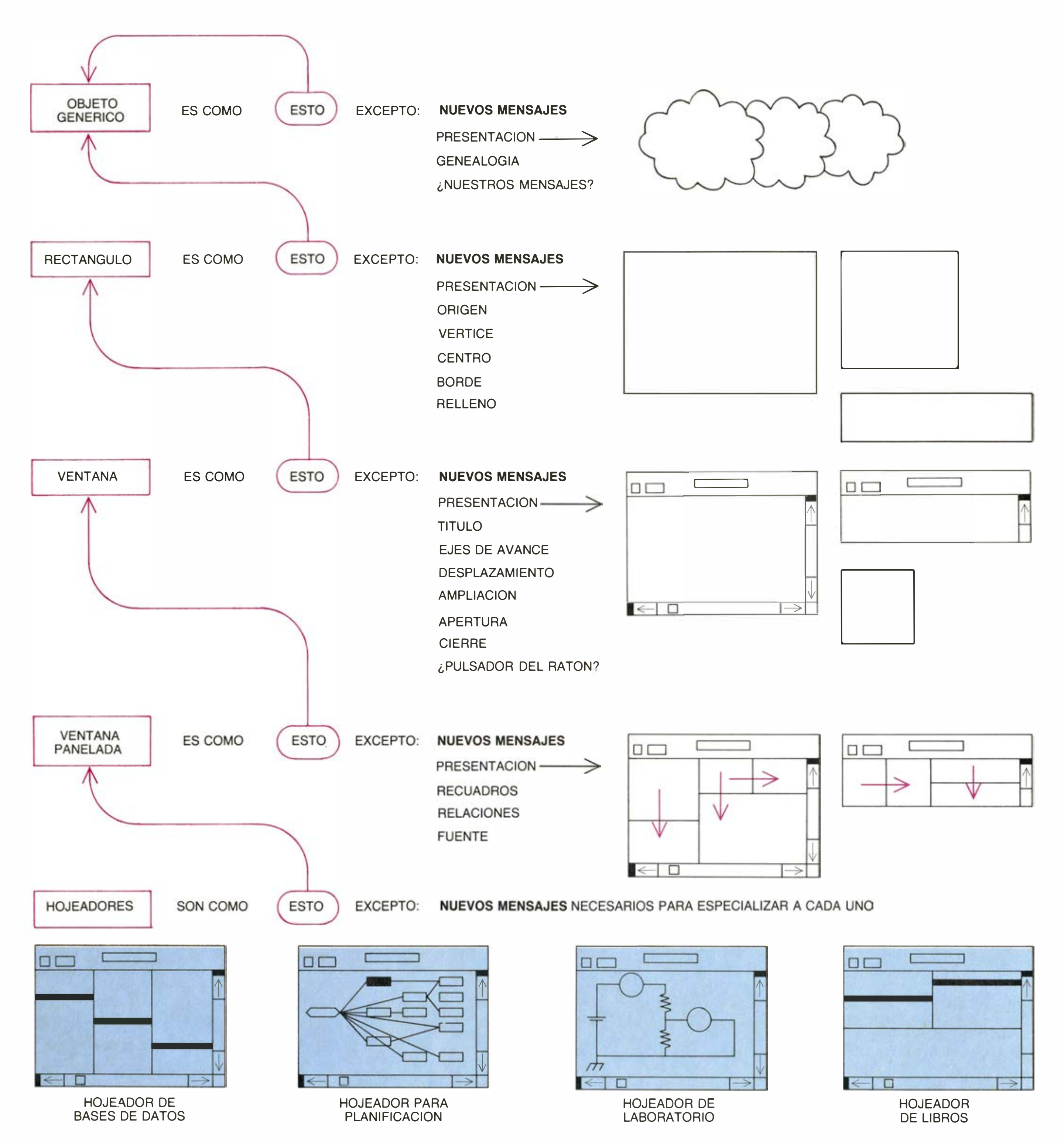
Muchos de los principios e ingenios que se han ideado para reforzar esta ilusión son hoy lugares comunes en el diseño de logicales. Tal vez el principio más importante sea el de que "se dispone de lo que se ve": las imágenes que muestra la pantalla son reproducción fiel de la ilusión del usuario. Al manipular la imagen presentada por ciertos procedimientos, se efectúa inmediatamente sobre el estado de la máquina alguna operación predictible (predictible en la forma en que el usuario imagina tal estado). Una ilusión hoy en boga consiste en crear sobre la pantalla "ventanas", "menús" e "iconos", que pueden seleccionarse mediante un indicador que permite señalar algunos de esos elementos. Al organizar en "ventanas" la pantalla de visualización cabe presentar a un tiempo cierto número de actividades. Se ofrecen al usuario "menús" de los pasos que es factible dar seguidamente. Los iconos representan objetos mediante imágenes concretas. Moviendo sobre la mesa un dispositivo indicador (comúnmente llamado "ratón") puede dirigirse sobre la pantalla un puntero que permite seleccionar ventanas concretas, elementos del menú o iconos.

Ha nacido así una nueva generación de programas interactivos, que buscan sacar provecho del efecto de ilusión del usuario. El objetivo es potenciar la capacidad de simulación del usuario. Las personas sacan el máximo partido de la máquina cuando tienen posibilidad de manejar su ilusión directamente, sin tener que recurrir a programas intermediarios, invisibles, pero necesarios para poner en funcionamiento hasta un sencillo procesador de textos. Lo que yo llamo aprovechamiento directo se alcanza cuando la ilusión actúa como herramienta o instrumental con el que resolver directamente un problema. El aprovechamiento será indirecto cuando la ilusión actúe como "agente", como una prolongación activa de los objetivos y metas de uno mismo. En ambos casos, el control de que dispone el diseñador de programas para "escenificar" lo que en esencia es una situación teatral, constituye la clave para crear una ilusión y reforzar el carácter "amistoso" con que se percibe.



2. LOS ESTILOS DE PROGRAMACION se suceden unos a otros a intervalos esporádicos, como aquí se ilustra con el ejemplo de algunos lenguajes de programación. Los lenguajes se han categorizado bastante arbitrariamente en niveles, aunque éstos (*bandas de color*) se superponen parcialmente. Hay lenguajes de bajo nivel (LBN), de alto nivel (LAN), lenguajes de muy alto nivel (LMAN) y de nivel ultra-alto (LUAN). Al ir evolucionando los lenguajes de programación, llegan a establecerse estilos (*tramos horizontales blancos*); años después se logra una mejora importante (*arcos blancos ascendentes*). Con el tiempo, el lenguaje perfeccionado deja de considerarse "mejora de lo antiguo" para constituir "algo casi totalmente nuevo", conducente a un género estable de nivel superior. El lenguaje Lisp ha evolucionado repetidamente.

Los primeros programas de ordenador fueron diseñados por matemáticos y científicos, quienes estaban convencidos de que la programación sería tarea lógica y directa. Los logicales resultaron más difíciles de modelar de lo que suponían; las computadoras eran máquinas testarudas, que se empeñaban en hacer lo que se les decía, y no lo que el programador quería decirles. Una nueva clase de artesanos tomó entonces a su cargo la tarea. Esos pilotos de pruebas del biplano binario no eran, a menudo, matemáticos, ni tan siquiera



3. PROGRAMACION "HEREDITARIA". Muestra la potencia de la descripción diferencial. Un objeto genérico (*arriba*) se presenta como una nube. A partir de ese objeto indiferenciado podemos formar un rectángulo, diciendo al efecto, "Quiero algo por el estilo, excepto...", y especificando después propiedades tales como la situación del origen (el ángulo superior izquierdo), la anchura, la altura y demás. Una elaboración de esta idea es la "ventana", una zona rectangular de la pantalla de visualización que proporciona una vista de la salida del programa. Al crear ventanas podemos hacerles heredar propiedades de los rectángulos que les sean aplicables, y añadir características nuevas, como pueden serlo "ejes de arrollamiento" (que permiten desplazar bajo la ventana el material que se está examinando), añadir un título y elementos de control, para modificar el tamaño y posición de la ventana. Puede crearse una ventana más compleja, definiendo en ella recuadros, o "paneles", y estableciendo comunicaciones entre ellos (*flechas de color*). Manipulando ventanas divididas en paneles se diseñan "hojeadores" ("*browsers*"): sistemas que permiten recuperar recursos y datos sin necesidad de nombrarlos.

científicos, pero estaban profundamente embebidos en una aventura romántica con el material (amores como éstos son muchas veces precursores del nacimiento de nuevas artes y ciencias). A las ciencias de la naturaleza les ha sido dado un universo, cuyas leyes es tarea suya descubrir. Las ciencias de cómputo crean leyes, dándoles la forma de programas, que los ordenadores se encargan de traducir, trayendo a la vida universos nuevos.

Programadores hubo que inhalaron demasiado profundamente los vahos embriagadores que exhala la idea de crear universos de carácter privado. Se convirtieron en lo que el eminente diseñador Robert S. Barton llamó "sumos sacerdotes de un culto rastrero". Empero, tarde o temprano casi todos descubrieron que una cosa es ser el dios de

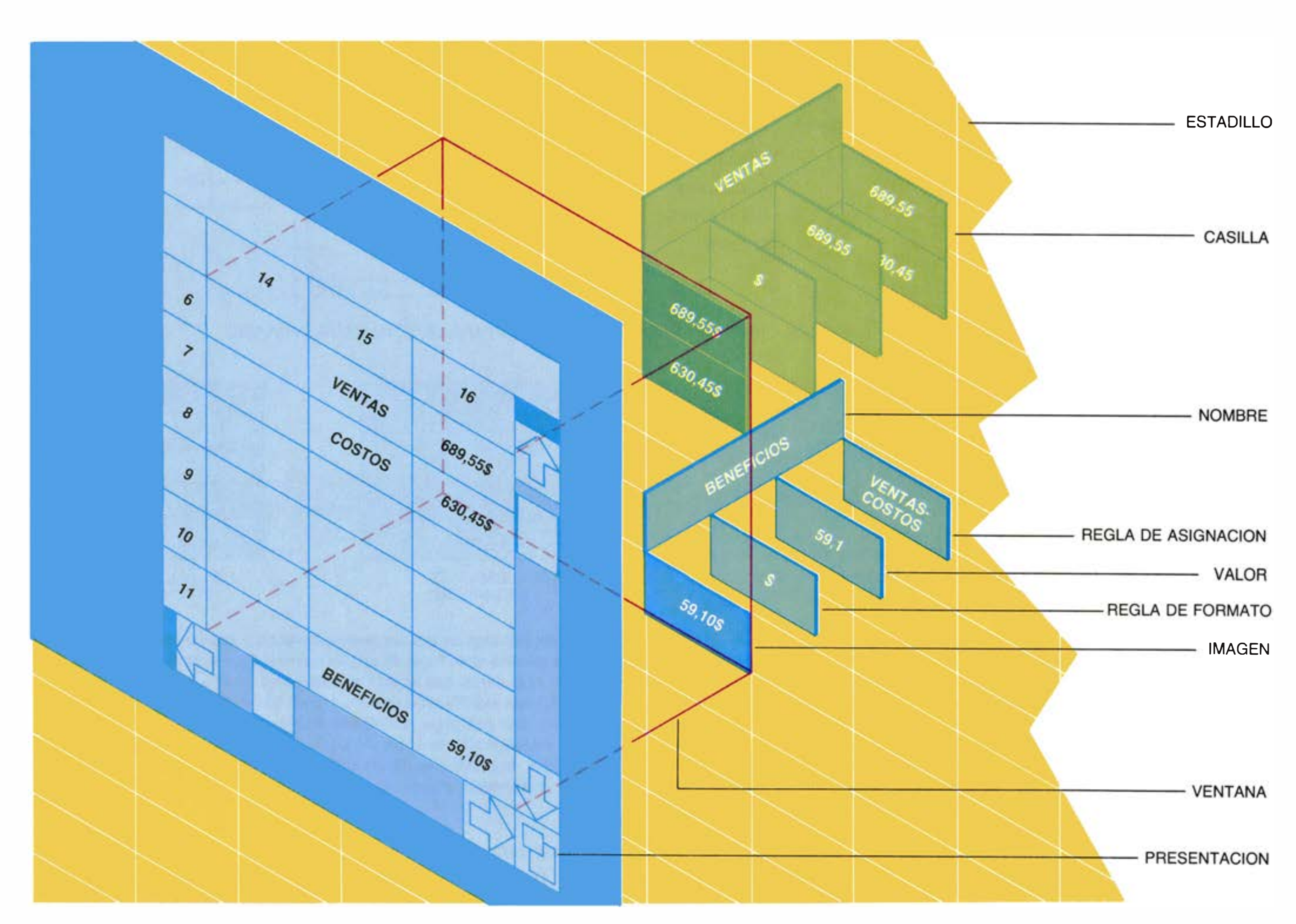
un universo y otra controlarlo. Tuvieron que mirar hacia campos distintos del suyo en búsqueda de inspiración e ideas para el diseño.

Un estilo vigoroso tanto puede darnos alas como dejarnos anclados. Las metáforas y símiles más traicioneros son los que parecen funcionar bien durante cierto tiempo, porque pueden impedir que nociones más vigorosas y profundas afloren y borboteen. En consecuencia, el progreso es lento, aunque algún progreso hay. Se establece un estilo nuevo. Algunos años después se realiza una mejora importante. Al cabo de algunos más, tal mejora no parece la "mejora de algo viejo", sino "algo casi nuevo" que conduce directamente a un nuevo estilo estable. No deja de llamar la atención la pervivencia de cosas antiguas y sus perfeccionamientos. Todavía prosperan hoy fuertes representantes de cada una de las eras pasadas, como el lenguaje llamado FORTRAN, que tiene 30 años, e incluso la vetusta grafía conocida por código directo de máquina. No faltarán quienes miren tales vestigios como fósiles vivientes; tampoco, quienes hagan notar que especies muy antiguas ocupan hoy nichos ecológicos específicos.

La informática no ha tenido todavía su Galileo o su Newton, ni su Bach o su Beethoven, ni su Shakespeare o Cervantes. Lo primero que necesita, ante todo, es un Guillermo de Occam, quien ya dijera "No deben multiplicarse sin necesidad las entidades". Mucho tuvo que ver la noción de que valía la pena dedicar considerables esfuerzos a suprimir la complejidad en favor de lo simple con el surgimiento de la matemática y la ciencia modernas, especialmente desde el punto de vista de la creación de una nueva estética, ingrediente vital de todo campo en activo desarrollo. Es una estética ajustada a las líneas de la cuchilla de Occam lo que en informática hace falta, tanto para enjuiciar la programación como para inspirar diseños futuros. ¿Cuántos son, exactamente, los conceptos que hay en este campo? ¿Cómo podríamos poner a nuestro servicio, buscando la reducción de la complejidad, ese maravilloso proceso que es la metáfora, el símil, la capacidad de encontrar semejanzas, e incluso identidades, entre estructuras?

En un estudio de 100 de sus más eminentes colegas, el matemático francés Jacques S. Hadamard encontró que la mayoría declaraba no valerse de símbolos en sus reflexiones, y que su enfoque de los problemas era fundamentalmente visual. Algunos, entre ellos Einstein, se remontaban aún más en su infancia, afirmando depender de "sensaciones de orden cinestésico o muscular". Las partes más antiguas del cerebro saben lo que hay que decir; las más nuevas, cómo decirlo. El mundo de lo simbólico sólo se maneja con eficacia cuando la agregación repetitiva de ejemplos llega a hacerse lo bastante aburrida para justificar su sustitución por una única noción abstracta.

La noción de variable, que permite representar y manejar una infinidad de casos concretos como si de una única idea se tratara, supuso en álgebra un avance impresionante. En el lenguaje, la metáfora acostumbra a acentuar y



4. ESTADILLO ELECTRONICO es un equipo de simulación, formado por agregación de objetos logicales llamados casillas, que pueden tomar valores unas de otras. La ventana permite seleccionar una porción rectangular de la hoja para su presentación en la pantalla. Podemos imaginar que cada casilla tiene tras la hoja varias capas, que calculan el valor de la casilla y determinan el formato de la presentación. El nombre de la casilla puede inscribirse en una de las contiguas. Cada casilla tiene asignada una regla que determina su valor, que tanto puede ser un número como una fórmula para calcularlo; puede hacerse que tal valor dependa de los estados de otras casillas de la hoja. La regla de formato confiere al valor la forma idónea de presentación.

destacar las semejanzas entre objetos, bastante o muy diferentes, como si en realidad fueran parecidos. Uno de los triunfos del pensamiento matemático fue la comprensión de que todavía más potentes podían ser determinados tipos de auto-comparación. El cálculo diferencial de Newton y Leibniz representa ideas complejas a base de encontrar formas de decir "Esta parte de la idea es como tal otra parte, excepto en que...". Los diseñadores de sistemas de cómputo han aprendido a hacer otro tanto con modelos diferenciales, por ejemplo con métodos de programación que gozan de propiedades llamadas hereditarias. En estos últimos años se han formulado modelos basados en la idea de recurrencia, en los cuales algunas de las partes son realmente el todo: es necesaria una representación del modelo completo para generar una representación de la parte. Entre los ejemplos tenemos la geometría de fractales, de Benoit B. Mandelbrot, donde cada subparte es similar a cualquier otra parte. Es el caos capturado en ley.

El diseño de partes con la potencia del todo es una técnica fundamental en la programación de hoy. Una de las más eficaces aplicaciones de la técnica está en el diseño de programas orientados a objetos. Se divide el ordenador (conceptualmente, aprovechando su capacidad de simulación) en cierto número de ordenadores más pequeños, llamados objetos, a cada uno de los cuales puede asignársele un papel, como a los actores en una obra. La tendencia hacia el diseño orientado a objetos sí representa una auténtica novedad y cambio de punto de vista –un cambio de paradigma– y trae consigo un enorme incremento de la capacidad expresiva. Se produjo un cambio similar cuando la eficacia, robustez y posibilidades energéticas de las cadenas moleculares que flotaban al azar en un océano prebiológico se vieron multiplicadas millones de veces al quedar encerradas en el interior de una membrana celular.

Las primeras aplicaciones de los objetos logicales se acometieron con los antiguos lenguajes secuenciales; los objetos funcionaban a modo de colonias de objetos unicelulares cooperativos. Empero, si buena era la idea celular, cuando la cooperación es lo bastante intensa para que las células se organicen en "supercélulas" –en tejidos y órganos– da comienzo lo verdaderamente interesante. ¿Podrá diseñarse la trama, indefinidamente maleable, en que está organizada la materia del ordenador, de suerte que forme un "superobjeto"?

Los estadillos dinámicos son buen ejemplo de este tipo de superobjetos "tisulares". Pueden considerarse equipos de instrumental de simulación, y proporcionan al usuario considerable grado de explotación directa. En sus formas más acabadas, los estadillos integran los estilos que llegaron a imponerse durante el decenio pasado (objetos, ventanas, disponibilidad de acceso directo al repertorio mostrado en pantalla, recuperación de datos por búsqueda orientada) poniendo así al día un producto antiguo, que verosímilmente va a ser uno de los productos "casi totalmente nuevos" de la corriente principal de los diseños venideros.

Los estadillos son agregados de objetos concurrentemente activos organizados por lo común en una matriz rectangular de casillas, parecida a las planas cuadriculadas que se usan en contabilidad. Cada casilla tiene asociada una regla o fórmula, que especifica cómo se determina el valor que ha de ocuparla. Cada vez que en algún punto del estadillo se cambia un valor, todos los valores dependientes de él se recalculan instantáneamente, presentándose en cada casilla los nuevos valores. Un estadillo es un universo simulado de bolsillo, que continuamente conserva su trama; es un instrumental válido para una sorprendente gama de aplicaciones. En su caso, la ilusión de usuario es sencilla, directa y vigorosa. No puede depararnos muchas sorpresas, porque el único modo de que una casilla tenga un valor es haciendo que se lo dé la regla particular correspondiente.

Los estadillos dinámicos fueron inventados por Daniel Bricklin y Robert Frankston, por la irritación que le producía a Bricklin tener que trabajar con los clásicos estadillos de papel pautado siendo estudiante de ciencias empresariales. Ambos quedaron sorprendidos ante el éxito de su idea y, más aún, porque casi todos los compradores del primer programa de estadillo (VisiCalc) lo utilizaran más para prever el futuro que para contabilizar el pasado. Tratando de hallar un corrector rápido crearon un instrumento de simulación.

Lograr que el estadillo electrónico se ponga a nuestro servicio es la sencillez misma. El símil visual amplifica la imagen que uno se forma de las situaciones y de las estrategias aplicables a ellas. La fácil transición del símil visual al método simbólico que determina el valor de cada casilla pone en juego toda la potencia de los modelos abstractos casi sin darnos cuenta. Una propiedad muy potente es su capacidad de convertir en genérica una solución, "pintando de un brochazo" una misma regla en docenas de casillas, sin exigir del usuario generalizaciones abstractas de su nivel inicial, concreto, de pensamiento.

El tipo más sencillo de regla de asignación convierte las casillas en objetos estáticos, como números o textos. Reglas más complejas pueden ser combinaciones aritméticas de los valores de otras cuadrículas, dadas a partir de sus posiciones absolutas o relativas o, lo que es preferible, de los nombres que se les asigne. Las reglas de asignación tienen capacidad de verificar condiciones y ajustar en consecuencia su propio valor. En versiones avanzadas cabe recuperar el valor de una casilla mediante búsqueda heurística orientada a un fin; de este modo, problemas para los que no existe método directo de solución pueden todavía intentar resolverse por un procedimiento de búsqueda.

La prueba más exigente de cualquier sistema no consiste en ver lo bien que se ajustan sus características a las necesidades previstas, sino en qué medida funciona cuando se quiera hacer algo que el programador no previó. Es menos cuestión de posibilidad que de perspicacia. Con otras palabras, ¿sabrá el usuario qué hacer, y hacerlo?

Supongamos que se quiera representar mediante un diagrama de barras verticales un conjunto de datos numéricos, barras cuya altura esté normalizada para el valor máximo; y supongamos que el sistema no tuviera prevista tal rutina. Incluso en lenguajes de programación de alto nivel ello requiere un programa fastidioso y desaliñado; con un estadillo se hace sin dificultad. Las casillas pueden actuar como elementos de imagen ("pixeles") de la presentación a realizar; un apilamiento de casillas forma una barra. En una barra que haya de alcanzar la tercera parte de la altura máxima, las casillas del tercio inferior han de ser negras y las de los dos tercios superiores, blancas. Cada casilla tiene que decidir si vestirá de negro o de blanco, según su posición en la barra: "Me presentaré de negro si mi puesto en la columna es menor que el dato a presentar; de lo contrario, iré de blanco" [*véase la figura 5*].

Otro ejemplo de estadillo es un refinado "hojeador" (*browser*) interactivo, sistema originalmente ideado por Lawrence G. Tesler, a la sazón en el Centro de Investigación de Xerox en Palo Alto. Se trata de un método muy cómodo para acceder a una base de datos organizada jerárquicamente, por selección sucesiva de listas. Se escribe primero desde el teclado el nombre de la base datos, que pasa a ocupar el encabezamiento de la pantalla; se recupe-

ran las áreas temáticas correspondientes a sus ramas inmediatas y se exponen en las casillas situadas bajo el encabezamiento. Puede elegirse una de las áreas temáticas señalándola con un "ratón"; el área seleccionada pasa entonces a ocupar el encabezamiento de la siguiente columna, lo que conlleva que, a su vez, se recuperen sus ramas por turno. Se prosigue de igual modo hasta alcanzar la información deseada [*véase la figura 6*]. Vale la pena destacar que el método completo puede programarse en el estadillo con sólo tres reglas.

No es la intención de este ejemplo hacer que todo el mundo abandone los trabajos de programación y se ponga a utilizar estadillos. Los estadillos actuales no alcanzan a tanto; ni probablemente la capacidad de simulación de la propia metáfora llegue a tanto. Empero, si por programar se entiende ir escribiendo recetas línea por línea, como se ha venido haciendo desde hace 40 años, entonces, esa actividad, que a muchos no interesaba, está seguramente obsoleta para todos. Los estadillos, y muy especialmente algunas de sus extensiones, como la explicada, son indicios vigorosos de que están a punto de ver la luz nuevos y más potentes estilos, tanto para noveles como para expertos. ¿Significa esto que la instrucción informática debe ser "de academia de conducir", que lo necesario será saber "conducir" programas, y que no será necesario aprender a programar? En absoluto. Los usuarios han de poder crear sistemas a su medida. Lo contrario sería tan absurdo como exigir que un artículo o ensayo tuviera que componerse ensamblando párrafos escritos de antemano.

Al comentar este medio, tan polifacético y proteico, he intentado hacer ver cuán eficazmente puede el diseño proporcionar facilidad de explotación al usuario, especialmente cuando el medio se ha moldeado como instrumento para la explotación y aprovechamiento directos. Está claro que al dar forma a los programas de aplicación son los creadores y usuarios quienes establecen las limitaciones al diseño, y no el medio. Quiero, no obstante, traer a primer plano el problema de las limitaciones de la programación, y manifestar mi opinión de que, en el futuro, formas nuevas y más potentes de explotación indirecta serán proporcionadas por agentes personales: extensiones de la voluntad y propósitos del usuario, que vendrán conformados por, e integrados en, la propia sustancia del ordenador. ¿Podrá el material originar mentalidad? Desde luego, no parece haber mucho de mental en una simple marca. ¿De qué modo podrá una combinación de marcas, incluso una combinación dinámica y reflexiva, mostrar por ningún concepto propiedades de mentalidad?

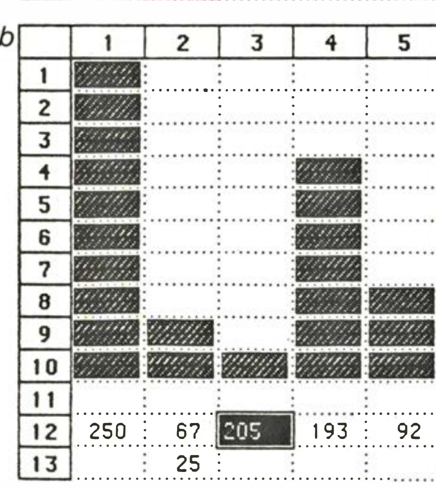
a

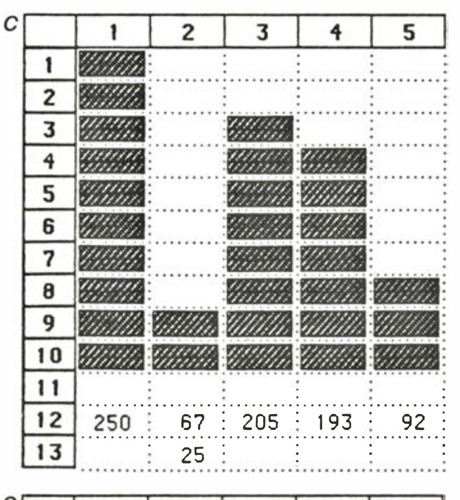
	1	2	3	4	5
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12	250	67	45	193	92
13		25			

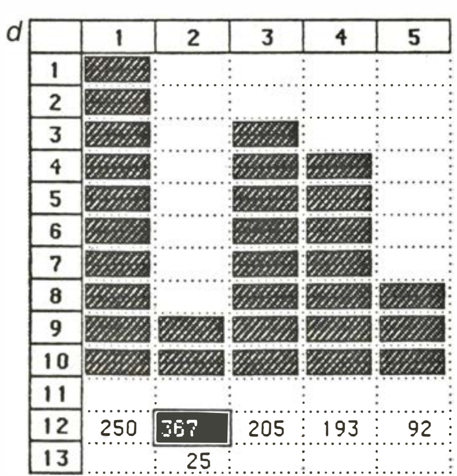
BARRA: La regla de asignación para cada casilla es "Casilla negra, si (11-posición vertical) × altura de cada casilla es menor que el dato (posición horizonal); si no, casilla blanca"

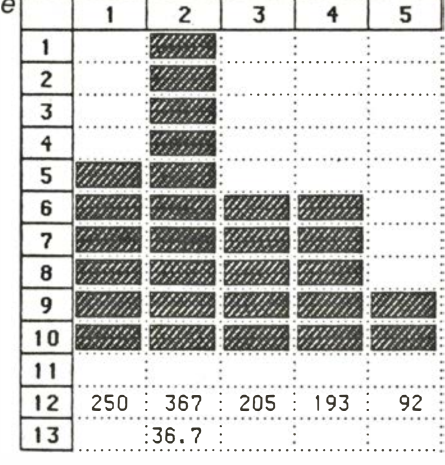
DATOS: La regla de asignación para cada casilla es, bien el propio número, bien un número que debe buscarse en algún otro lugar del estadillo.

ALTURA DE LA CASILLA: Máximo de los datos ÷ 10.









5. DIAGRAMA DE BARRAS obtenido sin dificultad a partir de los materiales ordinarios de un estadillo electrónico. Una barra corresponde a una columna de casillas, que actúan cada una de elemento de imagen ("pixel"). Una casilla asociada a cada columna contiene el dato, o valor, a representar por la altura de la correspondiente barra. Todas las casillas de una barra están gobernadas por una misma regla. La cantidad representada por la altura de un elemento de imagen es el valor del dato máximo a representar, dividido por el número de casillas que haya de asignarse a la barra más alta; en el gráfico *a*, cada barra contiene 10 elementos de imagen, y cada una de las casillas equivale a 25 unidades. Cada casilla será negra si su posición vertical en la barra, multiplicada por el número de unidades equivalentes a un pixel, es menor que el valor del dato correspondiente a esa barra; de lo contrario, será blanca. Cuando a una columna se le asigna un nuevo valor (*b*), aparece en esa columna una nueva barra (*c*). Si uno de los nuevos datos fuera mayor que el máximo de los anteriores (*d*), se redibujaría el sistema (*e*).

También los átomos parecen totalmente inofensivos. Mas la biología nos muestra que con materiales muy sencillos pueden alcanzarse organizaciones superlativamente complejas, capaces de interpretarse a sí mismas y de modificar dinámicamente su conducta. Algunas de ellas, ¡incluso parecen pensar! Por consiguiente, es imposible

negar ciertas posibilidades de mentalidad al material informático, pues justamente donde los programas llevan las mejores cartas es en la estructuración cinética de componentes simples. Los ordenadores "sólo pueden hacer aquello para lo que han sido programados"; pero otro tanto puede decirse de un óvulo fecundado en sus esfuerzos por convertirse en niño. Por otra parte, es imposible exagerar la dificultad de descubrir una arquitectura que genere mentalidad. Hubieron de transcurrir cientos de años de estudio en biología para que se descubrieran y elucidaran las propiedades del ADN y de su expresión, y se revelase que la célula viva era una arquitectura en progreso. Además, la biología molecular tiene la ventaja de estudiar un sistema ya ensamblado y en funcionamiento, mientras que para el compositor del soporte lógico, el ordenador es como una vasija de átomos que esperan integrarse en una arquitectura que el programador tiene que inventar y plasmar desde el exterior.

Insistiendo en la analogía biológica, muy poco puede decir la evolución a los genes acerca del mundo exterior, y menos, si cabe, pueden los genes decir al cerebro en desarrollo. Se han encontrado en el más de millón y medio de especies actuales todo tipo de niveles de competencia mental. La gama va desde conductas tan rígidas, tan "cableadas en firme", que en ellas aprender no es ni necesario ni posible, pasa por pautas conductuales que ha elaborado la experiencia, hasta un espectro de capacidades tan rico y polivalente, pero tan informe, que precisa una organización social estable –una cultura– para que se manifieste plenamente el potencial del individuo adulto. (En otras palabras, la forma que tienen los genes de hacer que los gatos cacen ratones es programar a los gatos para jugar; el resto se lo enseñarán los ratones.) Los investigadores en inteligencia artificial se han contentado, por lo general, con remedar el primer tipo de conductas, las cableadas en firme. Los frutos de sus esfuerzos suelen conocerse por "sistemas expertos". No es que sus inventores no sean honestos: pocos de ellos proclaman que sus sistemas hagan más de lo que pueden hacer, pero la etiqueta "expertos" evoca visiones que producen desilusión cuando al sistema se le escapa mucho de lo que constituye realmente una conducta experta (o cuando menos, competente), y de cómo una conducta llega a hacerse tal.

Tres desarrollos cuentan con pocas probabilidades de plasmarse en un futuro cercano. El primero es que pueda construirse una mentalidad humana adulta. El segundo es que se alcance una mentalidad infantil y se la "eduque" en un "ambiente" capaz de transformarla en mentalidad adulta. La tercera es que las actuales técnicas de inteligencia artificial contengan la simiente de una arquitectura a partir de la cual pudiera construirse una mentalidad genuinamente capaz de aprender hasta adquirir competencia. Que sus probabilidades sean muy reducidas no significa –y así es preciso subrayarlo– que la tarea sea imposible. Es verosímil que sea el tercer desarrollo el que primero se consiga. Pero antes de que así sea habrá sistemas que parezcan un tanto inteligentes, y como tales actúen; algunos pueden sernos verdaderamente útiles.

¿Qué aspecto tendrán los agentes en los próximos años? La noción de agente la ideó hace 30 años John McCarthy; el vocablo fue acuñado después por Oliver G. Selfridge, hallándose ambos en el Instituto de Tecnología de Massachusetts. Lo que ambos imaginaban era un sistema que, al dársele una meta a conseguir, pudiera tomar a su cargo los detalles de las oportunas operaciones del ordenador, y cuando se encontrase con dificultades, supiera pedir y admitir consejo, expresado en términos humanos. Un agente sería un "robot lógico", residente en el mundo del ordenador y encargado de gestionar sus asuntos.

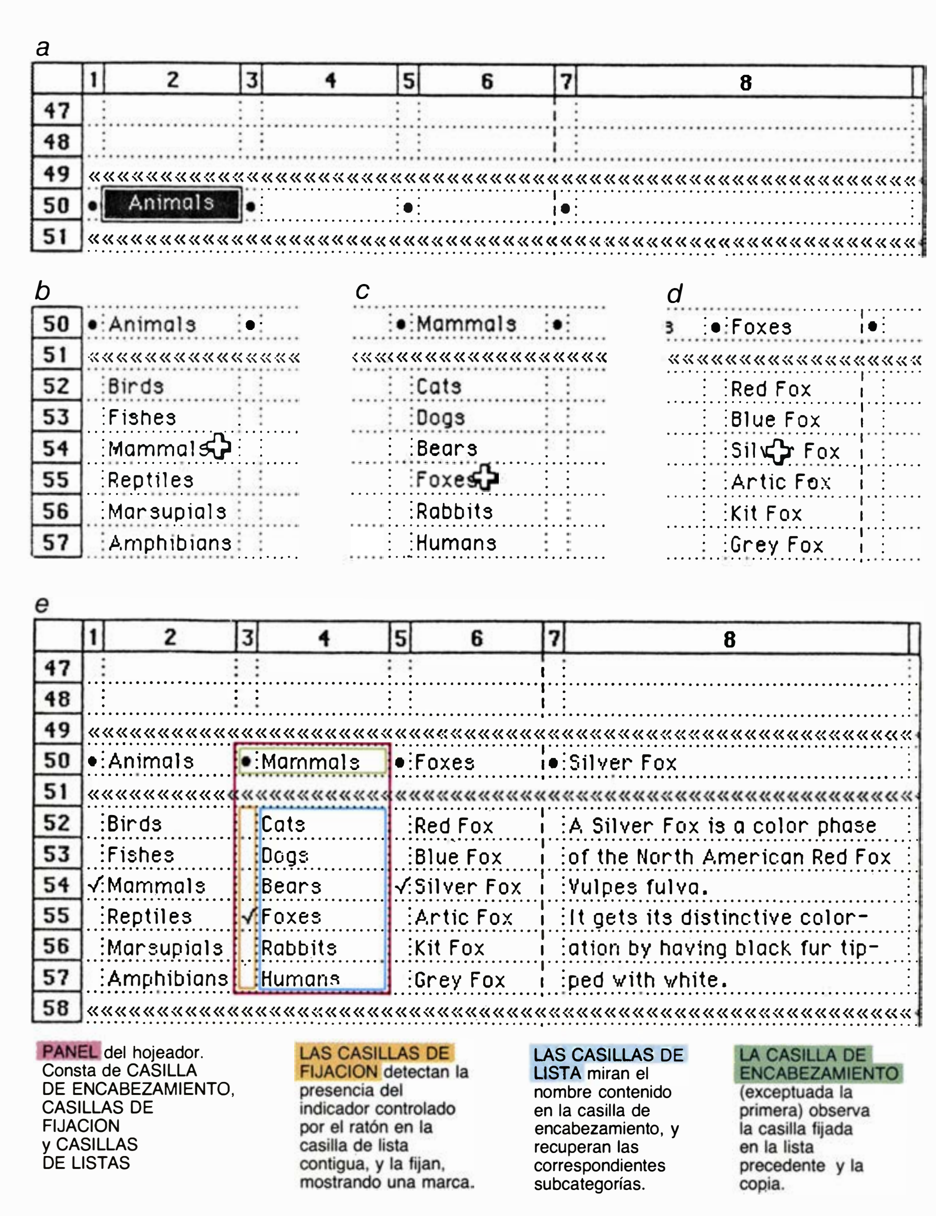
¿Qué funciones encomendar a un tal agente? Disponemos hoy, a través de las redes de ordenadores, de centenares de sistemas de almacenamiento y recuperación de informaciones. Es casi imposible conocer los arcanos procedimientos de acceso a cada sistema; una vez conseguido el acceso, el método de hojeado de los ficheros no es capaz de manejar mucho más de 5000 registros. Lo que nos haría falta es un agente que actuase a modo de bibliotecario, y nos ayudase a manejar la inmensa diversidad de posibles elecciones. Podría servirnos de piloto, capaz de ir trasladándose de una base de datos a otra. Todavía mejor sería un agente capaz de presentar ante el usuario todos los sistemas como un único y gran sistema, pero éste es problema de dificultad extraordinaria. Muy buena acogida tendría un "ratón de biblioteca" que, tenazmente, 24 horas al día, fuera recogiendo y buscando cosas que interesaran al usuario, y se las presentase condensadas, en una revista de carácter personal.

Aunque los agentes sean casi inevitablemente antropomórficos, no serán humanos, ni tampoco muy competentes, durante bastante tiempo. Violan muchos de los principios a satisfacer por una buena interfase de usuario. Sin duda éstos habrán de sentirse decepcionados si la ilusión que ante ellos se proyecta es la de una máquina inteligente, pero luego la realidad queda muy por debajo. Esa es la razón principal del fracaso de muchos diálogos sostenidos en lenguaje natural entre máquina y usuario, excepto cuando el contexto del diálogo está enormemente limitado para evitar las ambigüedades.

El contexto es la clave, desde luego. La ilusión de usuario es el teatro, el espejo definitivo. Es el público (el usuario) quien es inteligente y quien puede ser dirigido hacia determinado contexto. Darle a ese público el apunte, la entrada necesaria, es la esencia del diseño de buenas interfases con el usuario. Ventanas, menús, estadillos, etcétera, proporcionan un contexto en el cual la inteligencia del usuario es capaz de elegir correcta y sistemáticamente el paso siguiente. Un sistema basado en agentes tendría que lograr otro tanto; pero la creación de una interfase con algún semblante de mentalidad humana requeriría un enfoque mucho más sutil.

Cualquier sistema lo bastante poderoso para extender el alcance de las facultades humanas tiene también potencia suficiente para echar por tierra todo su mundo. Lograr que la magia del medio actúe en favor de los intereses y aspiraciones propios, y no en su contra, es, en lo que a ese medio concierne, salir del analfabetismo. En su acepción más simple, ser instruido, ser culto, significa fluidez y seguridad en el manejo del medio. No es bastante estar familiarizados, conocer la "gramática". No se puede decir de nadie que sea instruido porque sepa reconocer un libro y sus palabras, o una máquina de escribir y su teclado, o un ordenador y sus terminales de entrada y salida. No se ha salido del analfabetismo mientras no sea el contenido, y no la mecánica de la forma, lo que ocupe nuestro tiempo.

¿Es el ordenador un coche que conducir o es un ensayo que hay que pergeñar y redactar? Gran parte de la confusión al respecto se debe al intento de resolver la cuestión en este nivel. Tan proteica y mudable es la naturaleza del ordenador, que igual puede actuar como una máquina que como un lenguaje al que haya que moldear y sacar partido. Es un medio capaz de simular dinámicamente cualquier otro medio, incluidos medios que no pueden tener existencia material. No es una herramienta, aunque puede actuar como muchas de ellas. Es el metamedio primordial, capaz por ello de grados de



6. UN HOJEADOR DE BASES DE DATOS nos permite acceder a bases de datos jerárquicamente organizadas sin más que señalar elementos de listas sucesivas. Para saber del zorro plateado tecleamos "animales" en el primer recuadro (*a*). Aparecen entonces bloques temáticos de la base de datos llamada "animales" en el panel (*b*). Al seleccionar "mamíferos", tal subcategoría aparecerá en el panel siguiente (*c*); al elegir "zorros" se muestra su relación (*d*). Finalmente, obtenemos la descripción pedida (*e*).

libertad de representación y expresión jamás encontrados antes y, hasta hoy, sólo muy someramente investigados. Y más importante todavía, es ameno y divertido y, como tal, intrínsecamente interesante.

Si los ordenadores pueden ser coches, no cabe duda de que es deseable cierta instrucción informática, equivalente al saber conducir. En efecto, se está hoy intentando diseñar interfases de usuario que den acceso a la potencia del ordenador mediante interacciones más fáciles de aprender todavía que conducir automóviles. El papel y lápiz del futuro podrían muy bien ser programas integrados de procesamiento de textos, de grafismos, de simulación, de recuperación de informaciones y de comunicaciones de persona a persona. Empero, el nivel elemental de instrucción con lápiz y papel comienza ya en el jardín de infancia y en pre-escolar, y ello implica que la que podríamos llamar pre-escritura informática debería iniciarse lo más pronto posible; no debe esperarse a que los niños casi completen la enseñanza media para que reciban medio curso de esa instrucción, como proponen recientes informes de comisiones de educación en los Estados Unidos. (En España se están dando los primeros pasos para implantar enseñanzas de este tipo, con carácter optativo, en bachillerato y formación profesional, aparte cierto número de experiencias en centros de Educación General Básica.) Los niños precisan del equivalente informático de zapatos, bicicletas, autos y aviones desde el momento mismo en que comienzan a explorar el universo del conocimiento.

Pero, además, la instrucción con lápiz y papel no queda interrumpida en cuanto los niños aprenden a manipular un lápiz lo bastante para trazar ciertos símbolos sobre un papel. Una de las razones, es obvio, para enseñar a leer y escribir es que en el siglo xx tales conocimientos son necesarios en la vida cotidiana; otras hay, más críticas y de mayor vuelo. Con la lectura no sólo esperamos absorber los hechos concernientes a nuestra civilización y a las que nos precedieron; también dar con la estructura misma del estilo de pensamiento y de la imaginación. La escritura nos saca de las gradas y nos lleva al terreno de juego; el conocimiento, nuevo o antiguo, se hace verdaderamente nuestro al darle nosotros forma directamente.

En breve, estamos actuando como si saber leer y escribir fuera a servir para que la gente pensase mejor y de manera distinta. Hemos admitido que comenzar con el acervo de siglos de conocimiento de otras personas y otros pueblos es más eficaz que partir de la nada; que ello nos dará la base de lanzamiento para nuevas ideas. Admitimos igualmente que al expresar y conformar las ideas por medio de metáforas y otras formas de la retórica hacemos más plenamente nuestras esas ideas, y al mismo tiempo, potenciamos nuestra capacidad para aprender de otros. Sostenemos que todo ello es importante, a pesar de que leer y escribir parece ser muy difícil, y es tarea que requiere años. Nuestra sociedad declara que tal clase de instrucción no es un privilegio, sino un derecho; no una opción, sino un deber.

¿Y qué es entonces una formación en informática? No consiste en aprender a manejar un procesador de textos, un estadillo, o una de las modernas interfases con el usuario; tales habilidades siguen en el nivel del lápiz y el papel. Una cultura informática no es siquiera saber programar. Eso puede aprenderse en cualquier momento, y por métodos no más estimulantes que aprender gramática.

Una formación informática es un contacto con esa actividad lo bastante profundo para hacer fluido y grato el equivalente computacional de la lectura y la escritura. Como en todas las artes, puede muy bien admitirse una aventura romántica con el material. Si tanto valoramos el aprendizaje de artes y letras a lo largo de toda una vida como trampolín para el desarrollo y enriquecimiento personal y social, ¿deberemos dedicar esfuerzos menores a integrar la informática en nuestras vidas?

Algoritmos y estructuras de datos

Constituyen los elementos básicos de todo programa de ordenador. La elección de estructuras de datos y el diseño de procedimientos para manipularlas son decisivos a la hora de verificar que los programas cumplen con sus cometidos

Niklaus Wirth

Algoritmos y estructuras de datos son los materiales con que se construyen los programas. Es más, el propio ordenador consiste tan sólo en algoritmos y estructuras de datos. Las estructuras de datos a él incorporadas son los registros y las palabras de la memoria, donde se almacenan valores binarios; los algoritmos instalados ("cableados", se dice a veces) en el ordenador son reglas fijas, materializadas en circuitos lógicos electrónicos, gracias a las cuales los datos almacenados se interpretan como instrucciones a ejecutar. Así pues, en su nivel más fundamental, un ordenador sólo puede trabajar con una clase de datos, a saber, dígitos binarios, bits, y puede operar sobre los datos de conformidad con un único sistema de algoritmos, los definidos por el repertorio de instrucciones de su unidad central de proceso.

Raramente se expresan en bits los problemas que se decide resolver con ayuda de ordenador. Los datos se presentan, por lo común, en forma de números, caracteres, textos, hechos, símbolos y otras estructuras más elaboradas, como secuencias, listas o árboles gráficos. Mayor diversidad todavía muestran los algoritmos utilizados para resolver los problemas; de hecho, hay al menos tantos algoritmos como problemas abordables por computación. ¿Cómo puede una sola máquina resolver tan amplio espectro de problemas, si funciona siempre conforme a un sistema fijo de reglas? La explicación reside en que el ordenador es, en realidad, un ingenio de aplicación general, cuya naturaleza queda transformada por el programa que se le proporciona. El principio en que se funda lo enunció John von Neumann. En un momento dado, un caudal de información constituye un conjunto de datos procesados por el programa; momentos después, la misma información se interpreta por derecho propio como programa. En consecuencia, los programas se formulan mediante nociones familiares que convengan al problema que tengamos entre manos; posteriormente, otro programa, llamado ensamblador o compilador, traslada esas nociones a los medios y recursos disponibles en el ordenador.

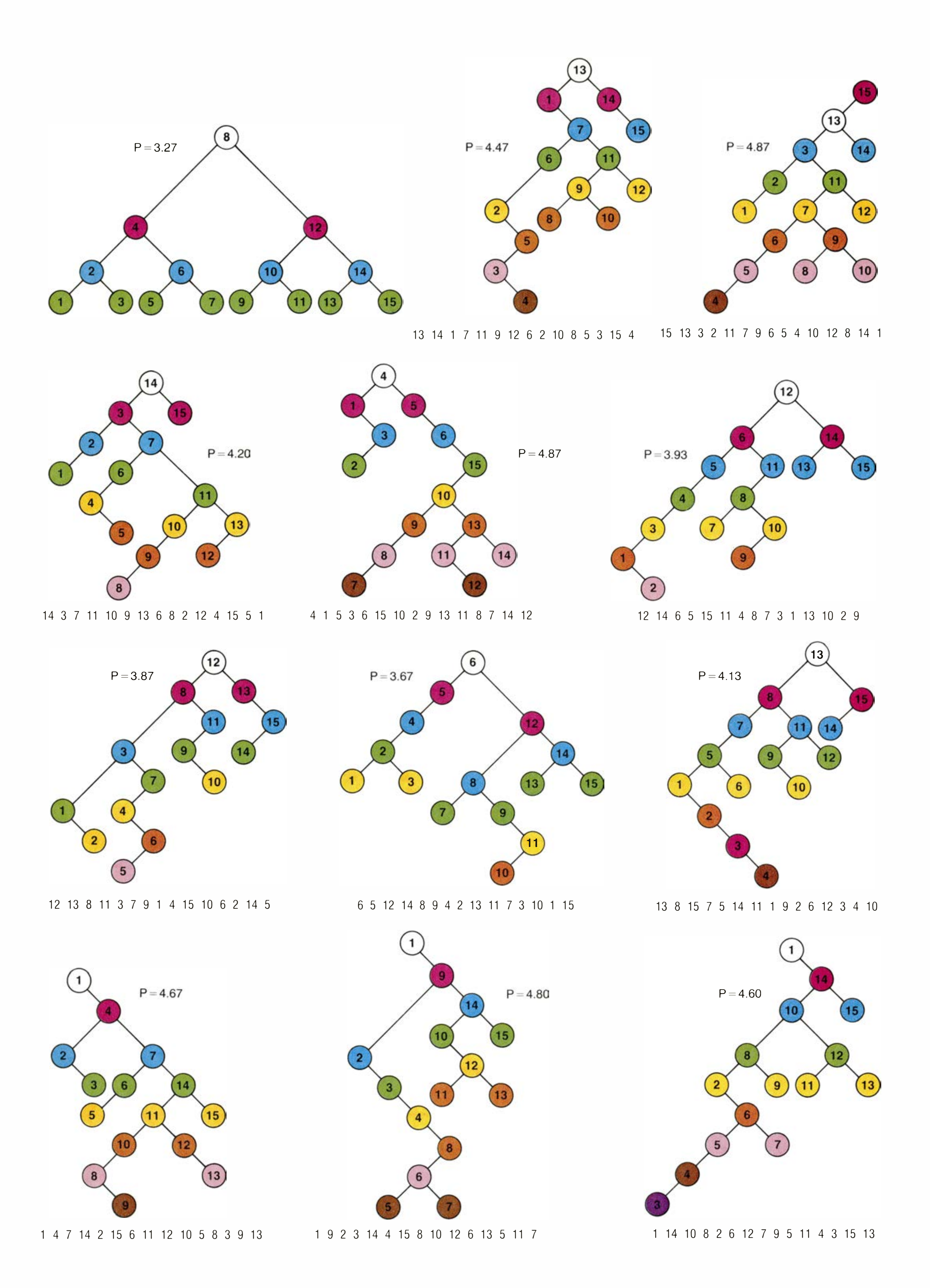
Podemos construir así sistemas de extraordinaria complejidad. El programador establece una jerarquía de abstracciones, esbozando primero el programa a grandes rasgos y atendiendo, después, a una sola parte cada vez, dejando de lado los detalles internos de las otras partes. El proceso de abstracción no es mera cuestión de conveniencia; es una necesidad, porque resultaría del todo imposible crear programas de tamaño no trivial si se tuviera que trabajar con una masa homogénea e indiferenciada de dígitos binarios. Sin estas abstracciones de más alto nivel, ni siquiera su propio creador alcanzaría a comprender del todo el programa.

Especificar en forma abstracta las estructuras de datos y los algoritmos de un programa requiere una notación formalizada, en la cual el significado de cada enunciado lícito esté definido unívocamente, con precisión y sin ambigüedad. Tales sistemas de notación formal han acabado llamándose lenguajes, no con mucha propiedad, pues la programación se asemeja sólo superficialmente a la escritura. Por mi parte, prefiero concebir la programación como la actividad que consiste en idear y proyectar una nueva máquina (cuyo funcionamiento se realiza por intermedio de otra máquina preexistente, de carácter universal). El diseño se especifica con los recursos que la notación nos proporciona, lo mismo que un aparato electrónico se diseña dibujando los símbolos de los componentes fundamentales del circuito y sus interconexiones. La necesidad de precisión resulta absolutamente evidente al comparar la redacción del programa con el diseño de una máquina.

Entre los recursos que casi todos los lenguajes de programación nos proporcionan se cuenta la posibilidad de referirnos a un elemento de datos asignándole un nombre, o identificador. Algunas de las cantidades nombradas son constantes, toman un mismo valor a lo



1. BOSQUE DE ARBOLES BINARIOS que pone de relieve la íntima interacción entre estructuras de datos y algoritmos. Un árbol binario, que por convenio crece descendiendo desde la raíz, es una estructura de datos elegida a menudo cuando es preciso recuperar aleatoriamente elementos integrados en un gran cuerpo de información. El árbol está compuesto por nodos, identificados por un valor clave; en los diagramas de la página adyacente, la clave es un entero comprendido entre 1 y 15. Cada nodo tiene, como máximo, dos "hijuelos", dispuestos de modo que la clave del izquierdo sea invariablemente menor que la de su progenitor, y que el derecho siempre la tenga mayor. El árbol óptimo es el del ángulo superior izquierdo: está perfectamente equilibrado, con lo que el número medio de nodos que es preciso atravesar hasta alcanzar uno dado es mínimo (la longitud del trayecto que conduce hasta cada nodo está indicado en color, de acuerdo con la clave superior). Los otros árboles se generaron mediante un algoritmo de inserción aleatoria, el cual añade un nodo en la primera posición que lícitamente pueda ocupar una clave, sin desplazar los otros nodos ni tratar de mantener equilibrado el árbol. Un algoritmo más refinado podría reducir un poco la longitud del trayecto medio, pero el algoritmo en sí sería más complicado. El algoritmo de inserción aleatoria y los beneficios de manejar árboles equilibrados se exploran en las figuras 9 y 10.



P = 3.27
P = 4.47
13 14 1 7 11 9 12 6 2 10 8 5 3 15 4
P = 4.87
15 13 3 2 11 7 9 6 5 4 10 12 8 14 1
P = 4.20
14 3 7 11 10 9 13 6 8 2 12 4 15 5 1
P = 4.87
4 1 5 3 6 15 10 2 9 13 11 8 7 14 12
P = 3.93
12 14 6 5 15 11 4 8 7 3 1 13 10 2 9
P = 3.87
12 13 8 11 3 7 9 1 4 15 10 6 2 14 5
P = 3.67
6 5 12 14 8 9 4 2 13 11 7 3 10 1 15
P = 4.13
13 8 15 7 5 14 11 1 9 2 6 12 3 4 10
P = 4.67
1 4 7 14 2 15 6 11 12 10 5 8 3 9 13
P = 4.80
1 9 2 3 14 4 15 8 10 12 6 13 5 11 7
P = 4.60
1 14 10 8 2 6 12 7 9 5 11 4 3 15 13

largo de todo el segmento de programa en que están definidas; por ejemplo, a *pi* podemos asignarle el valor 3,14159. Otras cantidades con nombre son variables, a las que se asigna nuevos valores mediante enunciados que forman parte del programa, con lo que su valor no se conoce hasta que se ejecuta el programa. Las variables *diámetro* y *circunferencia* pueden tomar nuevos valores cada vez que se efectúa un cálculo.

El nombre que se le dé a una variable o constante sirve de mnemónico al programador, pero carece de significado para la máquina. El compilador, encargado de traducir el texto de los programas en código binario, se limita a asignar a cada identificador una dirección en la memoria. Si una instrucción requiere multiplicar *diámetro* por *pi*, el ordenador toma cualesquiera números que se haya almacenado en las direcciones especificadas y calcula su producto; si el resultado ha de ser el nuevo valor de *circunferencia*, se almacena en la dirección de memoria correspondiente a esa etiqueta.

La designación de variables y constantes por medio de nombres tiene, en programación, un papel semejante al de la notación simbólica en álgebra; mas para que un ordenador maneje ese proceso es preciso suministrar cierta información adicional. Información que dé el "tipo" de cada una de las cantidades nombradas. Cuando una persona trabaja a mano en un problema, comprende intuitivamente de qué tipos son los datos y qué operaciones pueden efectuarse correctamente con cada tipo. Sabe, por ejemplo, que no se puede calcular la raíz cuadrada de un número negativo, ni escribir números con mayúsculas. Una de las razones por las que nos resulta fácil distinguir los tipos de datos es que los números, las palabras y los diversos símbolos tienen representaciones muy diferentes. En cambio, para el ordenador, todos los tipos de datos acaban descompuestos finalmente en una sucesión de dígitos binarios; y es forzoso explicitar las distinciones de tipo.

Imaginemos que, en el curso de alguna operación, uno de los registros de la unidad central de proceso de un ordenador esté cargado con el valor binario 1010011. ¿Cómo debe interpretarse ese valor? Una posibilidad es que represente un número cardinal (de los utilizados para contar), en cuyo caso, su equivalente, en notación decimal, sería 83. En muchos lenguajes de programación este valor binario podría también representar un entero con signo, en este caso, −45. El mismo dato binario podría codificar un carácter y no un número; en ASCII (American Standard Code for Information Interchange, código normalizado norteamericano para intercambio de información), el binario 1010011 especifica la letra *S*. Caben diversas posibilidades más. (De hecho, tal código binario podría ser una ins-

TIPO	FORMA DE PRESENTACION	REPRESENTACION INTERNA
CARDINAL	83	00000000 00000000 00000000 01010011 MAGNITUD
ENTERO	−83	1 1111111 1111111 1111111 10101101 MAGNITUD SIGNO
REAL	83.0	0 1000111 10100110 00000000 00000000 00000000 00000000 00000000 00000000 MANTISA NORMALIZADA EXPONENTE PONDERADO SIGNO DE LA MANTISA
CONJUNTO	0,1,4,6	00000000 00000000 00000000 01010011 ELEMENTOS DEL CONJUNTO
CARACTER	S	01010011 CODIGO ASCII
RISTRA	SET 83	S E T <ESPACIO> 8 3 00000110 01010011 01000101 01010100 00100000 00111000 00110011 CODIFICACION ASCII DE LOS CARACTERES NUMERO DE CARACTERES

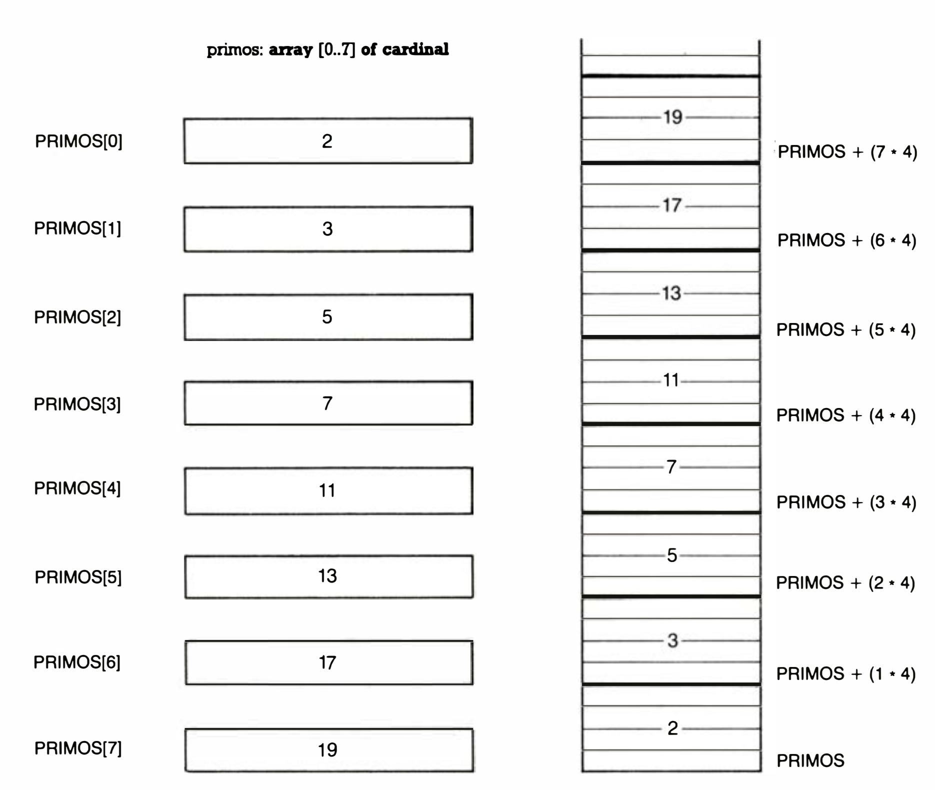
2. LOS TIPOS ELEMENTALES DE DATOS reciben del compilador una representación interna predeterminada; el compilador es un programa encargado de traducir a código binario los enunciados de un lenguaje de programación. Para que el compilador reserve espacio de almacenamiento y dé a los datos el formato preciso, debe especificarse los tipos de las variables. En las representaciones de datos mostradas aquí, a los números cardinales y a los enteros se les reservan 32 bits (cuatro octetos) a cada uno. A los números reales se les conceden 64 bits, y se almacenan en la llamada notación científica, que consta de una mantisa, un exponente y un bit para el signo. Un conjunto puede representarse mediante una cadena, o ristra, de dígitos binarios; un 1 denota que un elemento es miembro del conjunto, y un 0, que no lo es. A los caracteres se les asignan comúnmente 7 u 8 bits, de acuerdo con su valor en ASCII, un código estadounidense normalizado. Una cadena o ristra de caracteres consta de la secuencia de códigos individuales de los caracteres más un byte adicional que da la longitud de la serie. La descomposición en grupos se da aquí para facilitar la lectura; en el ordenador no hay separaciones.

trucción para el ordenador en vez de un dato; su interpretación dependería entonces del procesador concreto utilizado.)

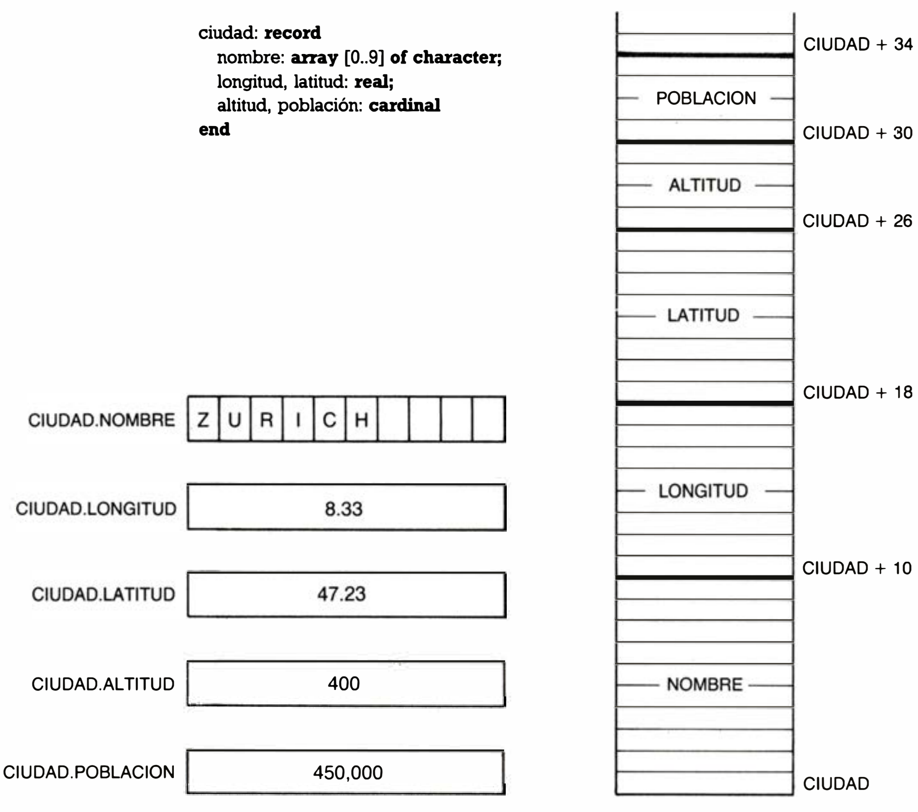
Entre los tipos de datos que los lenguajes de programación más habituales reconocen se cuentan los números cardinales (corrientemente llamados naturales, o enteros positivos), los números reales (dados mediante aproximaciones con decimales), conjuntos, caracteres y ristras (llamadas también "cadenas") de caracteres. La información acerca del tipo de cada variable no sólo se precisa para interpretar su representación binaria, sino también para reservar en la memoria la cantidad apropiada de espacio de almacenamiento. En muchos sistemas de cómputo modernos, a cada carácter individual se le asignan ocho *bits* de memoria (espacio conocido como "octeto" o *byte*), mientras que a los números cardinales y a los enteros pueden reservárseles dos o cuatro octetos, y a un número real hasta ocho octetos.

En algunos lenguajes de programación, el compilador infiere de qué tipo es una constante o una variable gracias a ciertos indicios dados por la forma en que viene escrita el valor que se le asigna; así, la presencia de un punto decimal podría indicar que se trata de un número real. En otros lenguajes, el programador tiene que enunciar explícitamente el tipo de cada variable. La declaración explícita de los tipos de los datos puede parecer fastidiosa cuando se dispone de métodos para evitarla, pero tiene una importante ventaja. Aunque el valor de la variable pueda cambiar repetidamente al ejecutarse un programa, su tipo no debería cambiar nunca; así pues, el compilador puede verificar que todas las operaciones efectuadas sobre la variable sean coherentes con la declaración del tipo. Tal comprobación de consistencia puede realizarse por examen del texto del programa, y, así, resulta válida para todos los posibles cómputos que el programa especifique. Por otra parte, una "ejecución de prueba" del programa compilado sólo podría certificar que el funcionamiento ha sido correcto para los valores específicos de las entradas que se han utilizado en el ensayo.

La noción de tipo de datos se ha generalizado (fundamentalmente en el lenguaje de programación Pascal) a fin de abarcar también la descripción de estructuras de datos. Las variables estructuradas constan de múltiples elementos (llamados componentes), pero



3. MATRIZ, O VARIABLE DIMENSIONAL de datos (también denominada *array*). Consta de un número especificado de elementos, todos ellos datos de un mismo tipo, lo que permite al compilador reservar la misma cantidad de espacio a cada elemento. Con ello, un elemento cualquiera puede localizarse mediante un sencillo cálculo que da su dirección a partir del valor de su índice. Tenemos aquí una matriz de ocho elementos que lleva el nombre *primos*, que constituye también la dirección de la memoria donde comienza la variable dimensional. Los elementos son números cardinales, cuyo almacenamiento requiere cuatro octetos por elemento; por tanto, la dirección de un elemento cualquiera de la matriz se calcula multiplicando por 4 el índice del elemento y sumando el número de octetos resultante a la dirección de *primos*.



4. ESTRUCTURA DEL REGISTRO que da cabida a informaciones heterogéneas. Aquí se ha definido un registro de nombre *ciudad* para almacenar cinco campos, a saber, una cadena de caracteres, dos números reales y dos cardinales. Para acceder a un campo determinado se da el nombre del registro y el nombre del campo, separados por un punto. Puesto que cada tipo de información exige un espacio de almacenamiento determinado, los campos no son todos de la misma longitud. Para cada campo, el compilador tiene que registrar un valor de desviación relativa (*offset*): la distancia que media, en la memoria, entre la dirección de comienzo del registro, considerado como un todo, y el comienzo del campo.

LISTA

```
type palabra =
  record
    ortografía: array [0..20] of character;
    siguiente: pointer to palabra
end
```

ANILLO

```
type día =
  record
    semana: (LUNES, MARTES, MIERCOLES, JUEVES, VIERNES,
      SABADO, DOMINGO)
    siguiente: pointer to día
end
```

ARBOL BINARIO

```
type primo =
  record
    número: cardinal
    izquierda, derecha: pointer to primo
end
```

5. ESTRUCTURAS DINAMICAS DE DATOS. Pueden dilatarse o contraerse, e incluso reorganizarse, bajo la dirección de la parte algorítmica del programa. Las estructuras están formadas por nodos (por lo común, registros), dotados de punteros, o indicadores (*color*), que señalan hacia otros nodos; un indicador que no señale hacia nada recibe el valor especial de *nil*. La estructura dinámica más sencilla es la lista concatenada, en la que cada nodo contiene un indicador que señala el consecutivo. El ejemplo que aquí se muestra podría formar parte de un diccionario. Se transforma una lista en un anillo haciendo que el último elemento señale hacia el primero. En los árboles binarios, cada nodo tiene dos indicadores.

puede aludirse a ellas como si fueran una entidad única. En un calendario, por ejemplo, hemos de poder especificar una fecha determinada, pero también debe haber un modo de aludir a meses o años completos. La declaración de tipo para variables estructuradas establece el número de elementos que la componen, permite al compilador asignarles el espacio de almacenamiento necesario y proporciona información sobre el método que se tiene intención de usar para acceder a ellos.

Si todos los elementos de una variable estructurada son de un mismo tipo (exigiendo cada uno el mismo espacio de almacenamiento) se dice que la variable es homogénea, por lo que cabe declararla variable dimensional, o matriz. (En inglés suele denominarse *array* a esas matrices. Emplearemos ese término cuando se trate de instrucciones de programa.) Así, una variable estructurada, a la que llamaremos *Sept*, y que consta de 30 números cardinales, podría tener la siguiente declaración de tipo:

Sept: array[1..30] de cardinal.

En una variable dimensional, o matriz, cada elemento individual queda identificado fácilmente por medio de un índice computado que representa un lugar dentro de la secuencia de elementos. Así, por ejemplo, la dirección del quinto elemento es, sencillamente, la dirección del primero, más cinco veces el tamaño de un elemento.

Tal facilidad de acceso se pierde cuando los elementos que componen una variable estructurada no son todos del mismo tipo. Por otra parte, los elementos que difieren en tipo probablemente difieren también en otros aspectos, por lo que es menor la necesidad de referirnos a ellos mediante índices calculados. Así, a cada elemento se le asigna su propio identificador. La variable estructurada completa así formada se llama variable de registro, o brevemente, registro. Sus elementos componentes se llaman campos. En la figura 4 vemos un registro ideado para recoger información sobre ciudades. Para aludir al registro se utiliza como nombre de variable *ciudad*; los campos individuales se designan *ciudad.nombre, ciudad.población*, etcétera.

Los conjuntos constituyen otro tipo fundamental de datos estructurados. Son de utilidad donde no sea de interés inmediato el valor de un elemento, sino que lo único que importe sea su presencia o ausencia. Si declara-

mos que una variable, de nombre *primos*, sea un conjunto de números cardinales, podemos definir una relación de pertenencia que genere el valor lógico "Verdadero" cuando un número sea miembro del conjunto *primos* y "Falso" en caso contrario. La creación y manipulación de conjuntos en el ordenador es muy sencilla; cada elemento se representa mediante un único *bit*, indicándose su presencia por un 1 y, su ausencia, por un 0.

Las variables dimensionales (arrays), los registros y los conjuntos constituyen las llamadas estructuras fundamentales. En muchos contextos hacen falta estructuras más complicadas, pero mejor que esforzarnos en inventar notaciones para todas ellas será preferible crear un instrumento general que permita construir estructuras arbitrarias. Mientras que la estructura de una variable dimensional, de un registro o de un conjunto, permanece invariable durante la ejecución de un programa, es posible permitir que las estructuras más elaboradas crezcan, se contraigan o modifiquen su topología. Las estructuras fundamentales son estáticas; las estructuras derivadas, dinámicas.

Dado que el tamaño de una estructura dinámica está sujeto a cambios, no podrá especificarse con una declaración invariante; tiene que definirse mediante la parte algorítmica del programa. Por idéntica razón, habrá de ser el programa, y no el compilador, quien deba encargarse de la asignación de espacio para la estructura. Esta situación pudiera resultar traicionera, pues no se podrá comprobar la corrección de la estructura durante la compilación. Hay, sin embargo, una propiedad de la estructura que sí permanece fija, y que puede, por tanto, declararse por anticipado, a saber, los tipos de los elementos de que la estructura se compone en último término. Durante la ejecución solamente puede variar el número de elementos y las conexiones entre ellos.

El mecanismo para crear estructuras dinámicas es un método de generación de componentes fundamentales, llamados nodos, y otro que establece conexiones entre nodos. Por lo general, los nodos son registros; las conexiones entre nodos están definidas por medio de variables llamadas indicadores, o también, punteros. Como su nombre sugiere, los punteros apuntan a elementos de la estructura dinámica; puede también asignárseles el valor *nil*, en cuyo caso no apuntan a nada.

Una de las estructuras dinámicas que más prestamente pueden crearse es la lista concatenada. Cada entrada, o

```
a := x; b := y; c := 0;
while b ≠ 0 do
    { aserto: a * b + c = x * y }
    { aserto: b ≠ 0 }
    b := b − 1; c := c + a
end
    { aserto: a * b + c = x * y y b = 0 da c = x * y }
```

OPERACION	EVALUACION	INVARIANTE DE BUCLE	CENTINELA
x := 7; y := 13			
a := 7; b := 13; c := 0	a = 7, b = 13, c = 0	7 • 13 + 0 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 12, c = 7	7 • 12 + 7 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 11, c = 14	7 • 11 + 14 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 10, c = 21	7 • 10 + 21 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 9, c = 28	7 • 9 + 28 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 8, c = 35	7 • 8 + 35 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 7, c = 42	7 • 7 + 42 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 6, c = 49	7 • 6 + 49 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 5, c = 56	7 • 5 + 56 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 4, c = 63	7 • 4 + 63 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 3, c = 70	7 • 3 + 70 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 2, c = 77	7 • 2 + 77 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 1, c = 84	7 • 1 + 84 = 7 • 13	b ≠ 0
b := b − 1; c := c + a	a = 7, b = 0, c = 91	7 • 0 + 91 = 7 • 13	b = 0

```
a := x; b := y; c := 0;
while b ≠ 0 do
    { aserto: a * b + c = x * y }
    { aserto: b ≠ 0 }
    c := c + a * (b MOD 10);
    a := 10 * a;
    b := b DIV 10
end
    { aserto: a * b + c = x * y y b = 0 da c = x * y }
```

OPERACION	EVALUACION	INVARIANTE DE BUCLE	CENTINELA
x := 7; y := 13			
a := 7; b := 13; c := 0	a = 7, b = 13, c = 0	7 • 13 + 0 = 7 • 13	b ≠ 0
c := c + a • (b MOD 10) a := 10 • a b := b DIV 10	c = 0 + 7 • 3 = 21 a = 10 • 7 = 70 b = 13 DIV 10 = 1	70 • 1 + 21 = 7 • 13	b ≠ 0
c := c + a • (b MOD 10) a := 10 • a b := b DIV 10	c = 21 + 70 • 1 = 91 a = 10 • 70 = 700 b = 1 DIV 10 = 0	700 • 0 + 91 = 7 • 13	b = 0

```
a := x; b := y; c := 0;
while b ≠ 0 do
    { aserto: a * b + c = x * y }
    { aserto: b ≠ 0 }
if Impar (b) then c := c + a end;
a := 2 • a;
b := b DIV 2
end
    { aserto: a * b + c = x * y y b = 0 da c = x * y }
```

OPERACION	EVALUACION	INVARIANTE DE BUCLE	CENTINELA
x := 7; y := 13			
a := 7; b := 13; c := 0	a = 7, b = 13, c = 0	7 • 13 + 0 = 7 • 13	b ≠ 0
if Impar (b) **then** c := c + a **end** a := 2 • a b := b DIV 2	c = 0 + 7 = 7 a = 2 • 7 = 14 b = 13 DIV 2 = 6	14 • 6 + 7 = 7 • 13	b ≠ 0
if Impar (b) **then** c := c + a **end** a := 2 • a b := b DIV 2	c = 7 a = 2 • 14 = 28 b = 6 DIV 2 = 3	28 • 3 + 7 = 7 • 13	b ≠ 0
if Impar (b) **then** c := c + a **end** a := 2 • a b := b DIV 2	c = 7 + 28 = 35 a = 2 • 28 = 56 b = 3 DIV 2 = 1	56 • 1 + 35 = 7 • 13	b ≠ 0
if Impar (b) **then** c := c + a **end** a := 2 • a b := b DIV 2	c = 35 + 56 = 91 a = 2 • 56 = 112 b = 1 DIV 2 = 0	112 • 0 + 91 = 7 • 13	b = 0

6. DESARROLLO DE UN ALGORITMO cuya finalidad es multiplicar dos números cardinales; contempla tres etapas. El primer algoritmo (*arriba*) se vale del método de adición reiterada; se puede verificar su validez por medio de una aserción, llamada invariante del bucle, que debe mantenerse verdadera en todas las fases del cálculo. El método utilizado que solemos aplicar para multiplicar a mano es más eficaz (*centro*): va dividiendo por 10 el multiplicador, en lugar de irlo decrementando de unidad en unidad. En el caso del producto de 7 por 13, el algoritmo rápido reduce de 13 a 3 el número de pasadas por el bucle. Dado que los ordenadores utilizan aritmética binaria y no decimal, un algoritmo basado en la división por 2 (*abajo*) es más rápido todavía, a pesar de que el número de operaciones que debe realizarse es superior.

asiento, de la lista es un registro, uno de cuyos campos es un puntero que señala cuál habrá de ser el siguiente registro. El puntero de la última entrada tiene el valor *nil*. Para añadir un registro a la lista, el programa provee el espacio de almacenamiento necesario y cambia el último indicador, a fin de que señale hacia el nuevo elemento de datos. Si se dispone que el último elemento indique otra vez hacia el primero, la lista queda convertida en un anillo. Para crear un árbol basta que cada nodo contenga indicadores que apunten hacia todos sus subnodos. Vemos en la figura 5 ejemplos de diversas estructuras dinámicas.

La creación práctica de los punteros no presenta dificultad: el valor de un puntero (salvo si es *nil*) es la dirección del nodo hacia el que apunta. ¿Por qué, pues, no llamar sencillamente direcciones a los punteros? Vale la pena mantener la distinción porque los punteros señalan, por estar así declarados, hacia un tipo de variable especificado, mientras que una dirección pudiera darnos una localización cualquiera de la memoria. De este modo, el compilador puede cerciorarse de que cada puntero está asociado a un objeto del tipo debido.

La noción de estructura de datos es, fundamentalmente, de carácter espacial; una estructura de datos puede quedar reducida a un mapa de cómo está organizada la información en la memoria de un ordenador. Los algoritmos son los elementos de procedimiento correspondientes; brevemente, las recetas para la computación.

Los primeros algoritmos se desarrollaron para resolver problemas numéricos, como la multiplicación de dos números, o el cálculo de su máximo común divisor, el cálculo de funciones trigonométricas, etcétera. En nuestros días, igual importancia tienen los algoritmos no numéricos; se han ideado para tareas como hallar el elemento mínimo de una secuencia, buscar en un texto una palabra dada, planificar o temporizar acontecimientos o clasificar datos según un orden especificado.

Los algoritmos no numéricos operan sobre datos que no son necesariamente números; además, para construirlos o comprenderlos no son precisas nociones matemáticas profundas. Mas ello no significa que en el estudio de tales algoritmos las matemáticas carezcan de lugar; por el contrario, son esenciales métodos matemáticos rigurosos para hallar las soluciones óptimas de problemas no numéricos, para demostrar la validez de sus resultados y para determinar su efectividad. La programación sigue siendo una disciplina fuertemente matemática. Las tornas, empero, han cambiado. En tiempos se empleaban métodos computacionales para resolver problemas matemáticos; ahora se están aplicando métodos matemáticos en la solución de problemas de cómputo.

No voy a intentar aquí dar una exposición general de las diversas categorías de algoritmos, ni tampoco analizaré métodos de construcción de nuevos algoritmos. Procuraré, en cambio, ilustrar el enfoque con que modernamente se razona sobre ellos. Dado un algoritmo, que se supone capaz de resolver un cierto problema, ¿cómo podemos comprenderlo? y, en particular, ¿cómo adquirir la confianza de que su funcionamiento es correcto sin recurrir a ensayarlo en un ordenador en unos cuantos casos, necesariamente particulares?

Considerado un algoritmo como una secuencia temporal de operaciones, es cuestión fundamental cómo está controlado tal flujo de operaciones. Cuando la ejecución de un programa ha alcanzado un enunciado concreto, ¿cómo puede saber el ordenador cuál es el enunciado a ejecutar seguidamente?

Se ha demostrado que tres principios de control bastan para describir cualquier algoritmo. El primero de ellos es

```
texto: array [0..M-1] of character
j =       0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
texto[j] = P E T E R   P I P E R     P  I  C  K  E  D     A     P  E  C  K

palabra: array [0..N-1] of character
i =        0 1 2 3
palabra[i] = P I C K

i := 0; j := 0;
while (i < N) and (j < M-N) do
  i := 0
  while (i < N) and palabra[i] = texto[j + i] do i := i + 1 end;
  if i < N then j := j + 1 end
end
```

```
P E T E R   P I P E R   P I C K E D   A   P E C K
P I C K
P I C K
  P I C K
    P I C K
      P I C K
        P I C K
          P I C K
            P I C K
            P I C K
            P I C K
              P I C K
                P I C K
                P I C K
                  P I C K
                    P I C K
                      P I C K
                        P I C K
                        P I C K
                        P I C K
                        P I C K
```

P: ∀i: (0 ≤ i < N): palabra[i] = texto[j + i]
Q: ∀k: (0 ≤ k < j): ∃i: { (0 ≤ i < N): palabra[i] } ≠ texto[k + i]
R: P y Q y (J ≥ m − n o i = n)

7. BÚSQUEDA DE UNA PALABRA EN UN TEXTO. Se efectúa mediante una serie de comparaciones, letra por letra. Texto y palabra se declaran como vectores ("arrays") de caracteres; en este caso, la palabra tiene cuatro letras y, el texto, 25. La primera letra de la palabra se compara con la primera letra del texto; se da la circunstancia de que coinciden (*aquí se indica por un círculo en color*). Se comparan, entonces, las segundas letras. Esta vez las letras difieren (*se indica con un círculo gris*). Se mueve, pues, la palabra un carácter hacia la derecha y se reanuda desde el principio de la palabra la comparación de pares de letras. Se habrá hallado la palabra cuando todas las letras concuerden. Las condiciones que ha de satisfacer el algoritmo se especifican en las tres proposiciones de cálculo de predicados que vemos en la parte inferior de la ilustración. El primer aserto (*P*) establece que, cuando la palabra está alineada en la posición *j* de la cadena de texto, para todos los valores de índice *i* se verifica que la letra *palabra*[*i*] es la misma que la letra *texto*[*j*+*i*]. La segunda proposición (*Q*) enuncia que ninguna sucesión concuerda exactamente para ningún valor de *j* que sea más pequeño. Así pues, lo que habrá de hallarse es únicamente la primera aparición de la palabra en el texto. La tercera proposición, que debe verificarse al finalizar la búsqueda, enuncia que *P* y *Q* se satisfarán ambas y, o bien *i* tendrá el valor *j* (lo que expresa que se ha encontrado una coincidencia en el lugar *j*), o bien *j* tendrá un valor más grande que el correspondiente a cualquier secuencia de coincidencia (indicando así que la palabra no está presente en el texto.)

tan obvio que, con frecuencia, se pasa por alto; es la noción de secuencia. A menos que al ordenador se le den instrucciones que lo eviten, irá ejecutando los enunciados del programa secuencialmente. El segundo principio es el de ejecución condicional que, por lo común, se designa en el texto mediante una construcción de tipo "si..., entonces..." ("*if... then...*"). En el enunciado "Si B, entonces E", la expresión B es booleana, esto es, una expresión que puede tomar, bien el valor de "Verdadera", bien de "Falsa", y E es un enunciado, o grupo de enunciados, cualquiera. Se evalúa B y, si es verdadera, se ejecuta E; de ser falsa, se salta la instrucción E.

El tercer principio es el de repetición, que puede venir indicado por una construcción "Mientras..., haz..." ("*While... do...*"). La instrucción "Mientras B, haz E" ("*While B do E*") va inspeccionando el valor de B; si este valor es "Verdadero", ejecuta E; los dos pasos se repiten hasta que, por primera vez, B produce el valor "Falso". En la mayoría de los casos, formando parte de E hay un enunciado que se encarga de que el valor de B acabe por cambiar, evitando así que el bucle se ejecute indefinidamente. Tanto en la construcción condicional como en la repetitiva, B tiene una función de "vigilancia"; es una expresión que sólo permite que E se ejecute mientras se satisfaga la condición definida por B.

Fijémonos en un algoritmo para multiplicar dos números cardinales, x e y, a base de adición reiterada. En la figura 6 se muestra una descripción formal del algoritmo. El primer paso consiste en definir tres variables: el multiplicando a, el multiplicador b y un total parcial, c, que acabará por ser el producto. Las variables reciben sus valores iniciales en tres enunciados $a := x$, $b := y$, $c := 0$. El símbolo ":=" aquí utilizado es un operador de asignación; mientras el signo igual constituye una aserción de igualdad, el operador de asignación crea realmente una condición de igualdad, es decir, asigna al símbolo situado en el primer miembro el valor de la expresión situada en un segundo miembro. Podemos leer la expresión de asignación como "Hagamos que a tome el valor x".

El corazón del algoritmo es un bucle de tipo "*mientras...*" en el cual la función de vigilancia está encomendada al enunciado $b \neq 0$. Mientras b sea mayor que 0, se efectuarán repetidamente dos operaciones. En la primera de ellas, b decrece en una unidad; en la segunda operación, se suma a al valor que en ese momento tenga c. Ambas operaciones quedan formalmente expresadas en dos enunciados de asignación:

$$b := b - 1; c := c + a$$

El propósito del algoritmo es evidente, y su ejecución, directa. Al ser c inicialmente igual a 0, lo que se hace es sumar a consigo mismo por b veces consecutivas, lo que satisface la definición directa del producto de a por b.

Sin embargo, ¿cómo estar seguros de que un programa escrito para expresar esta idea, intuitivamente clara, la ha encarnado realmente en el algoritmo correcto? Uno de los métodos consiste en cargar el programa en un ordenador, compilarlo y ponerlo a prueba en unos cuantos casos. Tal método jamás podrá proporcionar una confianza absoluta en el buen funcionamiento del programa, por la sencilla razón de que el número potencial de casos a ensayar es infinito. Una solución mejor consiste en incluir en el programa "asertos" o enunciados de condiciones que hayan de ser verdaderos (si el algoritmo está correctamente construido), independientemente de la trayectoria que haya seguido el cómputo para alcanzar ese punto. En el caso que nos ocupa, la aserción es un "invariante de bucle", esto es, un enunciado que se conserve verdadero por muchas veces que el bucle se ejecute.

¿Qué aserto relativo a las cantidades del problema de multiplicación se conservará verdadero a lo largo de toda la ejecución del programa? Dado que a y b se hacen al principio iguales a x y a y, respectivamente, está claro que, en el momento inicial, $a * b$ tiene que ser igual a $x * y$ (el asterisco significa multiplicación; este convenio es el adoptado en muchos lenguajes de programación). Análogamente, cuando el cálculo ha terminado, el valor de c se toma como producto y, por tanto, c tiene que ser igual a $x * y$. En las diversas etapas que van del comienzo al final del procedimiento, b decrece en una unidad cada vez que c se incrementa en a; de este análisis se sigue que, a lo largo de todo el cálculo, ha de verificarse la identidad $a * b + c = x * y$. Esta aserción es, por consiguiente, el invariante esencial del bucle "*mientras...*"; juntamente con la condición de finalización ($b = 0$), establece la validez del resultado deseado ($c = x * y$).

En este ejemplo, la veracidad de la aserción utilizada para confirmar que el programa está debidamente concebido apenas si es más evidente que la corrección de los propios enunciados del programa. Sin embargo, el algoritmo considerado en el ejemplo es muy simple. La potencia de los asertos como medio de verificar que los programas son correctos queda de manifiesto tan pronto refinemos el algoritmo, para hacerlo más eficaz. El aserto formulado para el caso más sencillo sigue siendo válido, incluso al ir ganando complejidad el programa.

El bucle del algoritmo de multiplicación por adición reiterada tiene que ejecutarse y veces. Existen métodos mucho más rápidos. El algoritmo de multiplicación que los niños aprenden en la escuela primaria es un buen ejemplo. Su principio es el mismo, pero la reducción de b es mucho más rápida. En lugar de decrementarse de 1 en 1, b se va dividiendo por 10, operación particularmente sencilla en el sistema decimal. Tanto es así, que en este contexto el proceso no acostumbra a considerarse de división, sino de mera descomposición del multiplicador en los dígitos que lo forman. La separación de los dígitos puede hacerse por métodos algorítmicos aplicando los operadores matemáticos DIV y MOD, que generan, respectivamente, la parte entera del cociente y el resto de la división entera.

La modificación del algoritmo para sacar partido de este procedimiento más eficaz puede guiarse directamente con la necesidad de conservar la invariancia del aserto $a * b + c = x * y$. El enunciado de asignación $b := b-1$ que figuraba en el algoritmo original se remplaza ahora por $b := b$ DIV 10; la invariancia requiere que a se multiplique por 10 y, por tanto, se añade el enunciado $a := a * 10$ al programa. Si b no es múltiplo de 10, habrá que restar de b el resto (b MOD 10) y, para mantener invariante la expresión, sumar a c el producto $a *$ (b MOD 10).

La elección del factor 10 en el procedimiento de multiplicación se debe a la facilidad de manejo del sistema de numeración decimal. Dado que internamente el ordenador se vale de números binarios, es ventajoso utilizar en lugar de 10 el divisor 2. Podría hacerse una sustitución directa del 10 por el 2, pero también caben ulteriores refinamientos. El algoritmo resultante es el utilizado en todos los ordenadores. La ganancia en eficiencia es importante, pues el número de iteraciones se rebaja desde y al logaritmo de y en base 2. De no ser por ese perfeccionamiento, los ordenadores se pasarían la mayor parte del tiempo multiplicando.

Podemos tomar un segundo ejemplo

en el dominio de los algoritmos no numéricos. Dado un texto almacenado como una sucesión de caracteres, la tarea consiste en hallar en su seno la primera aparición de una palabra, que puede definirse como una secuencia arbitraria de caracteres, de longitud no mayor que el texto. Los algoritmos construidos sobre ese modelo revisten importancia en muchos campos de la informática; su aplicación más obvia tal vez sea el tratamiento de textos.

El primer paso en la construcción del algoritmo consiste en especificar cuál es el resultado exacto que se desea. En la figura 7 tal especificación se ha formulado con una notación formal (el cálculo de predicados); aquí daré una mera descripción verbal. Las dos variables, el texto y la palabra, pueden declararse como vectores (arrays) de caracteres, con lo cual cualquier carácter que se elija puede recuperarse sin más que dar un índice calculado. Supongamos que el texto sea un vector de m caracteres, y la palabra, otro de n caracteres, siendo, por lo común, n mucho mayor que m.

¿Cuál será la condición cuyo cumplimiento quedará garantizado cuando se encuentre en el texto una sucesión de caracteres concordante con la palabra? Podemos formular la respuesta con auxilio de variables-índice i y j, que especifiquen, respectivamente, posiciones en los vectores (arrays) de palabra y de texto. Se da la concordancia si, para todos los valores de i, desde 0 hasta $n-1$ (es decir, para todo el intervalo de valores permisibles del índice), el carácter designado en el vector-palabra es el mismo que el carácter del vector-texto especificado por el valor índice $j + i$. El valor de j para el cual se cumple esta condición indica el primer carácter de la sucesión concordante, y puede servir como valor retornado por el algoritmo de búsqueda.

Pero encontrar una sucesión concordante no era todo lo pedido; el enunciado del problema pedía, además, que fuera la primera. Así pues, debe imponerse otra condición más en el algoritmo: que para todos los valores de índice j del vector-texto, menores que el valor en que comienza la secuencia concordante, el vector-palabra y el vector-texto deben diferenciarse, al menos, en un carácter. Si se satisfacen ambas condiciones, el resultado será válido.

Es preciso tener en cuenta otra posibilidad más: tal vez la palabra buscada no se encuentre en el texto. La omisión de casos como éste es causa frecuente de tropiezos en la programación. Podemos corregirlo especificando que, si no se encuentra la palabra, el valor de j tendría que ser mayor que el máximo posible para que desde él diera comienzo una secuencia concordante, a saber, $m - n$.

Las tres condiciones aquí definidas, esto es, que los caracteres de la palabra y del texto concuerden para todos los valores admisibles de i y de $j+i$, que no haya regiones de concordancia para valores de j más pequeños y que j sea menor o igual que $m-n$, representan asertos que pueden sernos útiles para verificar la corrección del funcionamiento de un algoritmo. Resulta que también son útiles como armazón o estructura sobre la que construir el propio algoritmo. En efecto, la vía más obvia para emprender la búsqueda es proceder por repetición. Se establece para ello un bucle "*mientras...*", donde la primera y la tercera condiciones actúan de centinelas y, la segunda, como invariante del bucle. Se hace igual a 0 el valor inicial de j, con lo que la búsqueda comenzará al principio del texto. En cada recorrido del bucle se comprueban las condiciones objeto de vigilancia y, si la palabra concuerda con el texto, o si j es mayor que $m-n$, el programa abandona el bucle; de lo contrario, j se incrementa en 1 y el bucle se recorre una vez más.

Queda por especificar cómo podrá detectarse la coincidencia propiamente dicha, es decir, cómo se compararán los caracteres del texto con los de la palabra sobre el intervalo de valores del índice que van desde $i=0$ hasta $i=n-1$. La solución consiste en un bucle encajado en el bucle primitivo; para cada valor de j, el bucle interior recorre el intervalo completo de valores de i y va comparando, de uno en uno, los n caracteres. En cuanto se produce una discrepancia, se abandona el bucle interior. El valor que tenga i al salir del bucle indicará si se ha encontrado o no la concordancia buscada: si i es menor que n, la comparación terminó prematuramente al haber encontrado una disparidad.

El algoritmo esbozado arriba es sencillo pero relativamente ineficaz. Lo que se hace, en esencia, es ir superponiendo el texto y la palabra, comenzando desde el principio de ambos, y compararlos carácter por carácter. Tan pronto se detecta una disparidad, la palabra se corre una posición hacia la derecha, y se repite la comparación. El proceso continúa hasta dar con una coincidencia total, o hasta que la palabra ha recorrido del todo el texto. Cuando el texto no contenga palabras concordantes, serán necesarias $m-n$ comparaciones, como mínimo. Por lo común, este número será bastante mayor.

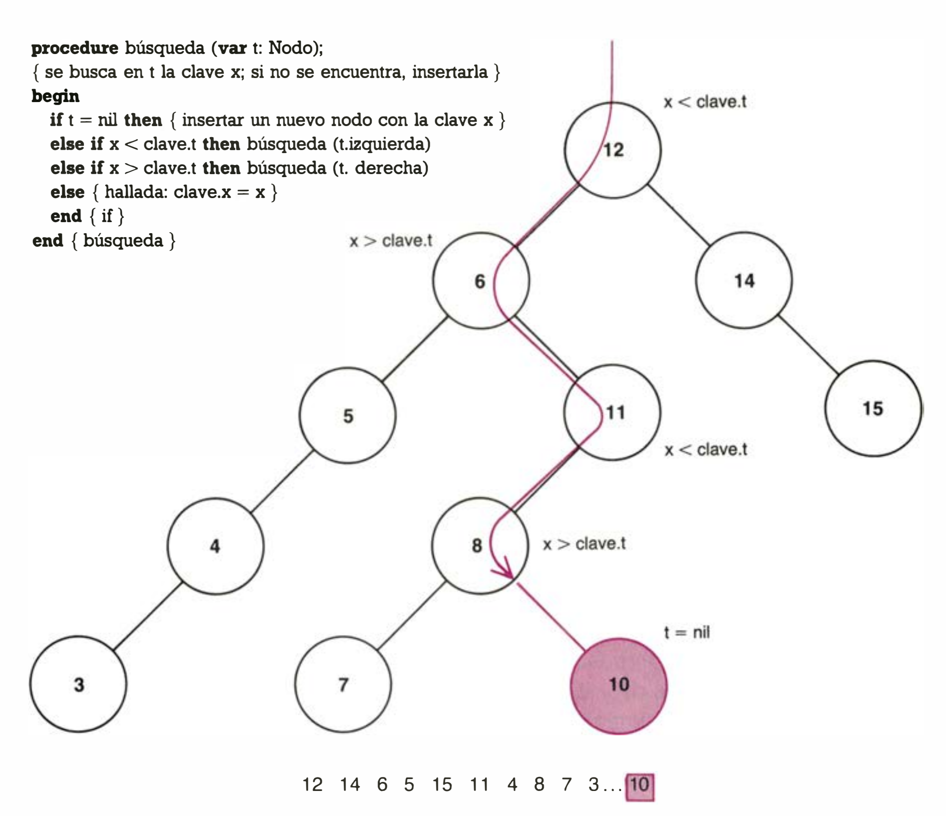
Siendo tan fundamental la tarea de

8. ALGORITMO DE BUSQUEDA MAS RAPIDO, ideado, en 1976, por Robert S. Boyer y J. Strother Moore, actualmente en la Universidad de Texas en Austin. La búsqueda comienza al principio del texto, pero en cada fase la comparación de letras se hace desde el final de la palabra. El programa ha de llevar una tabla que dé la distancia que media entre el fin de la palabra y la última aparición de cada letra en la palabra; cuando una letra no interviene en una palabra, se anota en la tabla la longitud total de la palabra. Cuando una letra de la palabra no casa con una letra del texto, se busca en la tabla el correspondiente asiento; la palabra se traslada hacia la derecha otros tantos caracteres. En la primera comparación hay en la palabra una *k* que no casa con la *e* del texto; por tanto, se hace avanzar la palabra dos espacios.

inspeccionar un texto, pudiera parecer inverosímil que, pasados ya 30 años de informática, fuera posible descubrir métodos notablemente mejores. Empero, en 1976, Robert S. Boyer y J. Strother Moore II, hoy en la Universidad de Texas en Austin, encontraron un método más rápido. Su idea permite que, en el bucle principal del programa, los incrementos de j sean mayores que 1. La comparación de la palabra con un segmento del texto se inicia al final de la palabra, retrogradando hacia el comienzo. Cuando una letra de la palabra deja de casar con la correspondiente del texto, se corre la palabra hacia adelante, para alinear con esta posición (que llamaremos posición pivote) la siguiente letra que sí case. Si la palabra no contiene ninguna letra que concuerde con la posición pivote, la palabra se mueve hacia delante de modo que su última letra se encuentre un espacio más allá del pivote.

Este procedimiento plantea inmediatamente la cuestión de cómo hallar el siguiente par de letras concordantes, pues si para ello es preciso comparar de uno en uno los caracteres, no se habrá ganado nada. Hay otro procedimiento: el programa puede mantener una tabla que enumere todas las distancias que hay entre el final de la palabra y la última aparición de cada letra en la palabra. Evidentemente, la compilación de la tabla requiere algún tiempo, pero su confección sólo tiene que hacerse una vez; si el texto es suficientemente largo, el esfuerzo vale la pena.

Tal vez el algoritmo de Boyer y Moore sea más rápido, ¿pero cómo tener la certeza de que está correctamente concebido? En particular, al desplazar varios lugares hacia la derecha la palabra, sin comprobación alguna, ¿podemos estar seguros de que no se ha pasado por alto ninguna alineación concordante? Un razonamiento informal es que un emparejamiento requiere la concordancia de todos los pares de letras, y los alineamientos sobrepasados difieren necesariamente en, al menos, una posición, a saber, la del pivote.



9. ALGORITMO DE INSERCION ALEATORIA, rutina cuya finalidad es mantener y actualizar un árbol binario; tanto puede añadir información como recuperarla. El algoritmo es un procedimiento recursivo que se aplica exactamente igual en cada nodo. Dado un valor clave x, el algoritmo lo compara con la clave del nodo que está examinando; si x es menor que la clave del nodo se explora la rama izquierda y, si es mayor, la derecha. De ser el valor de x igual al de la clave, se ha encontrado el nodo que se estaba buscando. Si la clave es *nil*, el nodo no existe; se crea, entonces, un nuevo nodo en esa posición del árbol. En el ejemplo mostrado se está explorando un árbol en búsqueda del valor clave $x = 10$ (*en color*).

Las principales preocupaciones de los programadores son la corrección y la eficacia de los programas. He mostrado ya que pueden, y deben, utilizarse métodos analíticos para establecer la correcta ideación de los programas, pues la verificación empírica exhaustiva exigiría demasiado tiempo, incluso en problemas sencillos. Precisamente por igual razón, la eficacia y el rendimiento no son empíricamente mensurables. El instrumento para analizarlos es el cálculo de probabilidades.

Supongamos que se nos da una lista de n números, de la que se nos pide hallar el máximo de sus valores. El método obvio es inspeccionar secuencialmente la lista, comparando cada elemento con el máximo de los valores hallado hasta ese momento. Podríamos declarar una variable, llamada *max*, y asignarle como valor inicial el del primer elemento de la lista. Cada uno de los elementos subsiguientes se va comparando con *max*, en un bucle de tipo "*mientras...*", y si el elemento es mayor, se le asigna a *max* el valor del elemento.

Es evidente que el bucle debe iterarse n veces, lo cual impone una cota inferior sobre el tiempo de ejecución del procedimiento. ¿Cuántas veces habrá de ejecutarse el enunciado de asignación? Si se diera la circunstancia de que el primer elemento fuera el mayor de todos, la asignación sólo se haría una vez; en cambio, si la sucesión fuese creciente, a *max* se le asignaría nuevo valor n veces. Así pues, 1 y n son los valores mínimo y máximo; pero, ¿cuál es el valor promedio? La cuestión mal puede responderse empíricamente. Hay $n!$ posibles ordenaciones de n números distintos, demasiadas para poderlas examinar todas, ni siquiera para valores pequeños de n. (Si el ordenador pudiese examinar un millón de esas ordenaciones por segundo, el examen exhaustivo de todas las listas de 16 números ordenados de todos los modos posibles exigiría medio año.)

El método analítico para la determinación del promedio es muy sencillo. La asignación inicial se ejecuta siempre una vez y, por tanto, el recuento de ejecuciones comienza en 1. Suponiendo equiprobables todas las ordenaciones, la probabilidad de que el segundo elemento sea mayor que el primero es 1/2; la probabilidad de que el tercero sea mayor que cualquiera de los dos primeros, 1/3. Prosiguiendo el análisis de este modo, resultará que el número promedio de asignaciones es igual a la suma $1+1/2 + 1/3 \ldots + 1/n$, la conocida serie armónica. En el siglo XVIII, el matemático suizo Leonhard Euler demostró que la suma de esta serie es aproximadamente igual al logaritmo natural de n, más una constante, llamada hoy constante de Euler, cuyo valor ronda en torno a 0,577. El logaritmo de n crece mucho más lentamente que el

propio n y, por tanto, el tiempo invertido en asignación de valores a *max* puede despreciarse frente al invertido en comparaciones e incrementación del índice del vector (array). Por consiguiente, es razonable afirmar que el esfuerzo requerido para hallar el máximo de n números es proporcional a n.

Pocos problemas prácticos ceden tan fácilmente, por lo que el análisis de algoritmos es campo de activa investigación. En muchos casos, basta saber cómo depende el tiempo de ejecución del tamaño del problema. El tiempo de ejecución podría, por ejemplo, ser directamente proporcional al tamaño del problema, o a su cuadrado, o crecer exponencialmente con él. De ser exponencial el crecimiento, el algoritmo es prácticamente inútil. La teoría que se ocupa de estas cuestiones es la del llamado "análisis de complejidad".

Podemos ilustrar con un ejemplo los métodos de análisis de complejidad: la construcción de un árbol binario, una estructura de datos que suele adoptarse cuando es importante la rápida recuperación de informaciones. Un árbol binario tiene dos notables propiedades: cada uno de sus nodos puede, como máximo, tener dos subnodos; las claves utilizadas para identificar los nodos se organizan de manera que, en cada nodo, la clave más pequeña se encuentre en el subárbol izquierdo. La búsqueda de una determinada clave puede realizarse muy eficazmente, pues en cada nodo basta una sola comparación para saber si se debe tomar la rama izquierda o la derecha; el número de posibilidades se reduce, así, a la mitad en cada nodo.

La eficacia del árbol es máxima cuando está perfectamente equilibrado, es decir, cuando cada nodo tiene exactamente dos subnodos. La longitud media de las trayectorias (el número esperado de comparaciones de claves) es entonces igual al logaritmo en base 2 del número de nodos. En el caso más desfavorable, cuando el árbol degenera en una simple lista por estar cada subnodo concatenado solamente con otro, la longitud media de la búsqueda es la mitad del número de nodos. De este resultado se deduce la importancia de cuidar que el crecimiento del árbol sea equilibrado. Empero, la conservación misma del equilibrio requiere un esfuerzo considerable, por lo que cabe preguntarse si tal esfuerzo adicional redundará en búsquedas más rápidas. Sorprendentemente, en muchos casos no es así.

Imaginemos que se leen en sucesión aleatoria n valores clave y se insertan en un árbol binario inicialmente vacío. La ordenación izquierda-derecha de las claves se va estableciendo al tiempo de irlas colocando, pero no se hace ningún esfuerzo por equilibrar el árbol. Se puede idear un procedimiento único que sirva tanto para añadir una nueva clave como para buscar una ya presente; el procedimiento se limita a encontrar el lugar del árbol que correspondería a la clave y, si está desocupado, insertarla en él. En la figura 9 se muestra un algoritmo diseñado con tal fin. Se trata de un procedimiento recursivo, que hace invocación de sí mismo. En cada nodo, el algoritmo realiza una de las siguientes cuatro posibles acciones. Si el valor de la clave allí situada es *nil*, se inserta la nueva clave. Si el valor de la clave ya situada en el nodo fuese igual al de la nueva clave, se emite un mensaje de éxito en la búsqueda. Si la nueva clave es menor que la encontrada, el procedimiento apela a una nueva copia de sí mismo para que examine el subnodo situado a la izquierda. Si fuese mayor, la búsqueda recursiva se encamina hacia el nodo de la derecha.

¿Cuál es la longitud media del camino en un árbol construido por este procedimiento de inserción aleatoria? Como antes, hay que descartar la medición empírica, pues hay $n!$ posibles secuencias de inserción; pero puede efectuarse una estimación probabilística. Supongamos que las claves sean los enteros de 1 a n, y que todas sus permutaciones sean equiprobables. Una cierta clave, que llamaremos i, será la primera en llegar. Se convierte así en la raíz del árbol. Cuando se haya insertado el resto de las claves, habrá $i-1$ nodos a la izquierda de la raíz, y $n-i$ nodos a su derecha. Si se diera la casualidad de que i fuese el valor central de la gama, el árbol estaría perfectamente equilibrado en este nivel máximo; si i es igual a 1 o a n, las primeras ramas estarán totalmente desequilibradas.

La idea crucial del análisis es que puede aplicarse recursivamente el mismo razonamiento a cada subárbol. Si la segunda clave en llegar es j, y j es menor que i, entonces, cuando el árbol quede lleno, habrá $j-1$ nodos en la rama situada a la izquierda de j e $i-j$ en la situada a su derecha. Es más, a partir de esta descripción recursiva del árbol podemos calcular la longitud media del camino, gracias también a otro procedimiento recursivo. En la raíz, la longitud del camino es 1 (por la propia raíz), más la longitud de un subárbol con $i-1$ nodos, más la longitud de un subárbol que contiene $n-i$ nodos. Estas cantidades no se conocen pero pueden calcularse aplicando el mismo procedimiento al siguiente nivel del árbol. Finalmente se alcanzará el extremo de cada rama, donde la longitud del camino es, para cada nodo, bien igual a 0, bien a 1.

$P_n \approx \log_2 n$

$P_n = n/2$

i

i−1

n−i

$$P_n(i) = [(i-1)(P_{i-1}+1) + 1 + (n-i)(P_{n-i}+1)]/n$$

$$P_n = 2(n+1)[1 + 1/2 + 1/3 + 1/4 \ldots]/(n-3)$$

$$P_n = 2 \ln n - 1.845$$

10. AL EQUILIBRAR UN ARBOL BINARIO solamente se obtiene, en el caso general, un moderado aumento de la eficacia. En un árbol binario totalmente equilibrado, con n nodos (*arriba, a la izquierda*) el trayecto promedio tiene longitud proporcional al logaritmo de n. En el caso más desfavorable, el de un árbol degenerado en lista (*arriba, a la derecha*), la longitud del trayecto medio es $n/2$. Para árboles construidos por inserción aleatoria, la longitud media tiene que determinarse por análisis probabilístico. Si la raíz del árbol es el elemento i, el subárbol situado a su izquierda habrá de contener $i-1$ nodos y, el situado a su derecha, $n-i$. La longitud del trayecto promedio total es igual a la suma de las longitudes de los dos subárboles, más uno, por el nodo raíz. El análisis puede repetirse con cada subárbol, hasta las hojas, donde cada subárbol tiene una longitud de trayectoria igual a 0 o a 1. En el aleatorio, la longitud media del trayecto es función logarítmica de n, un 38 % mayor que la función del árbol equilibrado.

La definición recursiva de la longitud del camino tiene que promediarse para todos los posibles valores de i, desde 1 hasta n. El resultado, que se muestra en la figura 10, es una expresión que, nuevamente, hace intervenir la serie armónica. La longitud media de un árbol construido sin prestar atención al equilibrio es función logarítmica del número de nodos, y difiere de la correspondiente al árbol óptimo en un factor constante. El caso promedio está mucho más cercano del óptimo que del pésimo; es un 38 por ciento más largo.

Aunque los programas que he comentado aquí no son triviales, son muy cortos. En las aplicaciones prácticas, los programas tienden a ser largos e intrincados, y parecen crecer tan rápidamente como lo consiente la capacidad de memoria de los ordenadores en que han de funcionar. ¿Podrán aplicarse a tales programas los métodos de análisis esbozados? El autor tiene la convicción de que así es, pues los sistemas complejos exigen con mayor urgencia aún que los sencillos un razonamiento exacto.

La mayoría de los grandes sistemas de programación (sistemas de *software*) se basan en unos pocos algoritmos "profundos", en general construidos a partir de algoritmos básicos, como los de multiplicación y búsqueda, que aparecen en muchas variantes y combinaciones. Por otra parte, las estructuras de datos tienden a ser superlativamente complejas. En consecuencia, la elección de la representación correcta de los datos tiene, con frecuencia, mayor peso en el funcionamiento del programa que los detalles del algoritmo utilizado, siendo no pocas veces la clave del éxito en programación.

Es improbable que llegue a existir una teoría general que determine la elección de la estructura de datos. Lo mejor que se puede hacer es comprender perfectamente los bloques constructivos fundamentales y las estructuras construidas a partir de ellos. La capacidad de aplicar ese conocimiento a la construcción de sistemas grandes es, sobre todo, cuestión de destreza ingenieril y experiencia. Para adquirir tal destreza, el programador deberá luchar constantemente contra la complejidad, negarse a usar métodos que no comprenda totalmente y no renunciar nunca a la búsqueda de soluciones más sencillas y elegantes. En este esfuerzo, ninguna de las modernas herramientas de ingeniería del soporte lógico puede remplazar la facultad de razonar de manera precisa y constructiva del programador.

Lenguajes de programación

Ofrecen múltiples maneras de especificar una computación. El lenguaje transforma el ordenador en una "máquina virtual" cuyas características y capacidades las determina la programación

Lawrence G. Tesler

Un lenguaje de programación es algo más que una notación para dar instrucciones a una computadora: un lenguaje y el soporte lógico (software) que lo "entiende" pueden hacer de la máquina otra, transformándola en una nueva dotada de un carácter enteramente distinto. El material (hardware) de un ordenador arquetípico lo componen los registros, celdas de memoria, sumadores y demás; cuando un programador escribe en el lenguaje nativo de un computador, ésas son las componentes que debe tener presentes. Cada nuevo lenguaje lleva consigo un nuevo modelo de máquina. Aunque el soporte material no haya cambiado, el programador puede pensar en términos de variables, no en celdas de memoria, en ficheros de datos, no en canales de entrada y salida, y en fórmulas algebraicas, en vez de registros y sumadores. Bastan unos cuantos lenguajes para conferir al ordenador una personalidad dividida: se convierte en un haz de oficinas independientes que realizan sus propios cálculos y se envían mensajes entre sí.

Los lenguajes de programación y sus dialectos se cuentan por centenares, si no por millares. Los lenguajes naturales de comuniación humana quizá les superen en número, pero en determinados aspectos los lenguajes de programación divergen todavía más. Cada lenguaje tiene su gramática y sintaxis distintivas, su manera de expresar las ideas. En principio, muchas tareas de cómputo podrían llevarse a cabo con cualquier lenguaje, pero los programas se manifestarían muy diferentes; además, escribir un programa para una tarea dada sería más fácil en algunos lenguajes que en otros. Describiré aquí algunos lenguajes de programación de uso común y abordaré, sin entrar en detalles, los elementos que todos ellos comparten y las particularidades que los distinguen.

La figura 2 ilustra varias fases del desarrollo de un corto programa en Logo, un lenguaje inventado a finales de los sesenta por Seymour Papert y sus colegas en el Instituto de Tecnología de Massachusetts. Posee el Logo una característica interesante: su habilidad en el control de la "tortuga", un pequeño dispositivo robótico que se mueve adelante y atrás, gira sobre sí mismo y sube o baja una pluma que va pintando la traza de la marcha que sigue el reptil. En muchos casos, la tortuga real se substituye por otra simulada en la pantalla de vídeo.

La versión inicial del programa consiste solamente en comandos dados a la tortuga. Se empieza bajando la pluma (*pendown*); a continuación, los dos comandos *forward 50* (adelante) y *right 144* (derecha) se repiten cinco veces; termínase subiendo la pluma (*penup*). En su cumplimiento de las instrucciones, la tortuga dibuja una estrella de cinco puntas. El comando *forward 50* insta el trazado de una línea recta de 50 unidades de longitud; *right 144* especifica una vuelta dextrógira de 144 grados, el cambio de orientación que ocurre en cada vértice de una estrella regular de cinco puntas.

Si la escritura de una lista de comandos a ejecutar secuencialmente fuera el único método de expresar nuestras intenciones al ordenador, la creación de un programa complejo resultaría punto menos que imposible. En realidad, Logo y los demás lenguajes de programación aportan múltiples recursos para simplificar y generalizar las instrucciones. En este caso concreto, la parte de programa más visiblemente necesitada de mejora es la repetición quíntuple del par de comandos *forward* y *right*. Siempre que puede, el programador evita escribir algo más de una vez; y no sólo porque le resulte pesado teclear. Si el programa pudiera condensarse, ocuparía menos espacio en memoria. Más aún, la repetición aumenta la probabilidad de errores tipográficos, en particular cuando se está revisando el programa. Podemos ahorrarnos la repetición reemplazando los cinco comandos de movimiento de la tortuga con la sentencia *repeat 5* (repetir) [*forward 50 right 144*].

Supongamos ahora que el programador desea dibujar una estrella de nueve puntas cuyos lados midan 80 unidades de longitud. Podría realizarse por medio de una sentencia de la forma *repeat 9* [*forward 80 right 160*]; pero es obvio que está duplicando la misma estructura básica de programa, con meras diferencias de detalle. Mejor solución sería definir un procedimiento general donde el número de puntas y la longitud del lado se den como cantidades variables. En Logo, la palabra *to* (para, hacia) introduce una definición de procedimiento. Por tanto, la expresión *to estrella* indica que las instrucciones que siguen deberían almacenarse en calidad de método para dibujar una estrella. En adelante, *estrella* constituirá un nuevo comando del lenguaje, que podrá introducirse en un programa, lo mismo que sucedió con los comandos incorporados *forward* y *right*.

En el procedimiento *estrella*, las variables pertenecen a la clase llamada parámetros, que se "pasan" al procedimiento en cuanto se le invoca. En Logo, el nombre de un parámetro va precedido por el signo ortográfico dos puntos. Así pues, nuestro procedimiento se definiría con una frase del siguiente tenor: *to estrella :tamaño :puntas*. Tecleando *estrella 80 9* se asignará un valor de 80 al parámetro *tamaño* y un valor de 9 a *puntas*; se genera de este modo una estrella de nueve puntas con un lado de 80 unidades.

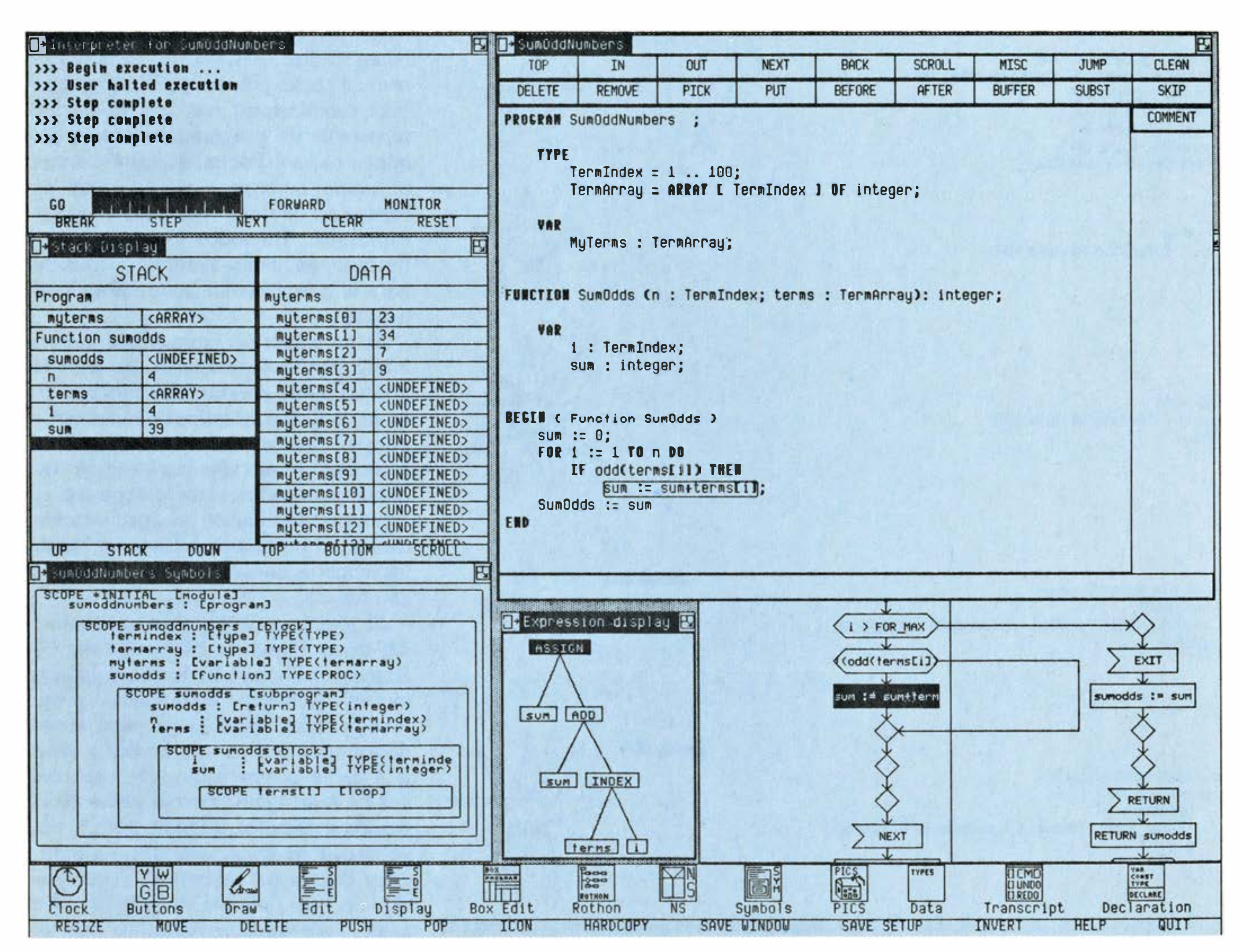
Podríamos perfilar un poco más el procedimiento *estrella*. En Logo, cualquier procedimiento puede reclamarse por el programador o por otro procedimiento. Observamos aquí una razón de potencia importante, pero aumenta el

peligro de que se pasen parámetros inapropiados a un procedimiento. Por ejemplo, en la maraña de un programa podría pasar inadvertido que se pidiera a la tortuga dibujar una estrella con sólo una o dos puntas. El problema puede atacarse añadiendo la cláusula *if puntas > 2* al programa. La cláusula *if* (si) sirve de voz de prevención que deja dibujar a la tortuga sólo cuando el número de puntas especificadas es mayor que dos.

Del ejemplo anterior se desprende que el Logo guarda, cuando menos, un parecido superficial con los lenguajes naturales. Tiene un vocabulario de palabras, números y otros signos (que se agrupan bajo la denominación común de "símbolos") que pueden ensartarse juntos para formar construcciones más largas, a modo de frases. Algunos de los símbolos (o *tokens*, por recordar el anglicismo) son palabras clave, con un significado fijo; otros los define el programador. Algunos símbolos desempeñan funciones verbales; otros, sustantivas; y los hay que hacen la función de modificadores o signos de puntuación. Las reglas que gobiernan las posibilidades de combinación de los símbolos constituyen una gramática.

Las creaciones de un lenguaje de programación suelen clasificarse en declaraciones y mandatos. Una declaración define lo que una cosa es, lo que significa o cómo está estructurada. En el programa Logo, *to estrella :tamaño :puntas* es una declaración que otorga a *estrella* el nombre de un procedimiento y define *tamaño* y *puntas* como variables de las que *estrella* se sirve cual parámetros. Por otro lado, el mandato describe generalmente parte de un algoritmo; especifica la acción a acometer. En muchos casos, el mandato adquiere forma imperativa: empieza con un verbo, al que sigue un objeto o un modificador. En la sentencia *repeat*, se trata de repetir (que es un verbo), el número de veces subsiguientemente indicado, que cumple funciones adverbiales y el material encerrado entre corchetes constituye el complemento del verbo.

Aunque el vocabulario y la sintaxis del procedimiento *estrella* son peculiares del Logo, los mecanismos que controlan el flujo de la ejecución a través del procedimiento se encuentran en la mayoría de los lenguajes de programación. En ausencia de cualquier sentencia de control explícita, la ejecución se produce de una forma secuencial. En el supuesto caso de que el ordenador esté leyendo el programa, leerá las líneas de arriba abajo, de suerte que, si no se altera el flujo, cada mandato lo ejecutará una vez.



1. INSTANTANEA DE UN PROGRAMA en ejecución, ofrecida con el sistema de desarrollo de programas PECAN. Una buena parte del texto fuente, o programa original, se exhibe en la gran ventana superior de la derecha; se trata de un programa en lenguaje Pascal para sumar los números impares de un vector de enteros. Los comandos que controlan la ejecución del programa se dan en la ventana superior izquierda. Se ha detenido la ejecución en la orden donde verdaderamente se lleva a cabo la suma; dicha sentencia aparece encuadrada en la ventana del texto fuente. Debajo de éste se ve parte de un organigrama del programa; a la izquierda del mismo, hay un árbol binario que muestra la estructura lógica de la orden de asignación. Las cajas anidadas o recuadros, abajo a la izquierda, indican el alcance de los símbolos del programa. La ventana titulada stack-display (*arriba, izquierda*) da una visión de las estructuras de datos del programa. PECAN fue desarrollado por Steven P. Reiss, de la Brown University, autor, asimismo, de esta ilustración.

Un elemento del programa que altera el decurso de la ejecución es el mandato *repeat*, que constituye un ejemplo de bucle o construcción iterativa. Cuando la computadora encuentra *repeat n* seguido por un grupo de mandatos encerrados entre corchetes, lee y ejecuta *n* veces el material demandado. Otro modo de controlar el progreso del computador a través de un programa es la ejecución condicional, que en el Logo (como en otros muchos lenguajes) la materializa la orden *if*. Los mandatos precedidos por un *if* se ejecutan sólo si se cumple la condición especificada.

En torno a las ideas básicas de ejecución iterativa y condicional giran muchas variaciones. La orden *repeat* es útil si se conoce, antes de introducirse el bucle, cuántas veces hay que ejecutarlo. Hay otras construcciones que permiten que la finalización del bucle la controlen sucesos internos del propio bucle. En varios lenguajes un mandato que empieza con la palabra clave *while* (mientras) se repetirá en tanto que persista una determinada condición. Otro modo de desviar el flujo de un programa es a través de un "salto incondicional" u orden *goto* (ir a), que desplaza la ejecución hacia un nuevo punto del programa. En los últimos años se ha ido desacreditando la orden *goto* entre los expertos en programación, ya que los programas con muchos saltos son difíciles de seguir (por el lector humano, no por el ordenador).

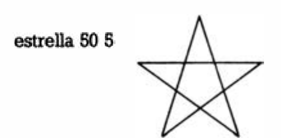
La noción de definición de procedimiento es un elemento vital en la programación. Constituye el mecanismo principal de abstracción, el proceso que convierte ejemplos específicos (una estrella de cinco puntas de 50 unidades de lado) en conceptos generales (una estrella de cualquier número de puntas y de cualquier tamaño). El procedimiento se define sólo una vez y sólo una vez se almacena en memoria; ahora bien, a lo largo del programa puede reclamarse desde muchos lugares, consiguiéndose que el producto de una elaboración mental se aproveche en repetidas ocasiones. Siempre que se llama a un procedimiento, se transfiere la ejecución al área de memoria donde está almacenado; llegado el procedimiento a su fin, la ejecución se reanuda en la instrucción inmediatamente siguiente a la llamada. Una clase especializada de procedimiento, la función, devuelve un valor de cierto tipo al programa instante. Por ejemplo, la función tangente toma un ángulo como parámetro al ser llamada y devuelve un valor igual a la tangente del ángulo.

```
pendown (pluma bajada)
forward (adelante) 50 right (derecha) 144
forward 50 right 144
forward 50 right 144
forward 50 right 144
forward 50 right 144
penup (levantar la pluma)
```

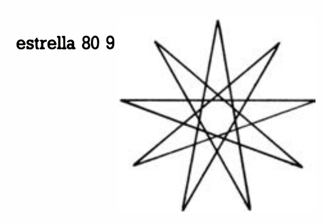
```
pendown
repeat 5 [forward 50 right 144]
penup
```

```
pendown
repeat 9 [forward 80 right 160]
penup
```

```
to estrella :tamaño :puntas
  pendown
  repeat :puntas [forward :tamaño right 720/:puntas]
  penup
```




```
to estrella :tamaño :puntas
  if :puntas <2
    [pendown
    repeat :puntas [forward :tamaño right 720/:puntas]
    penup]
```

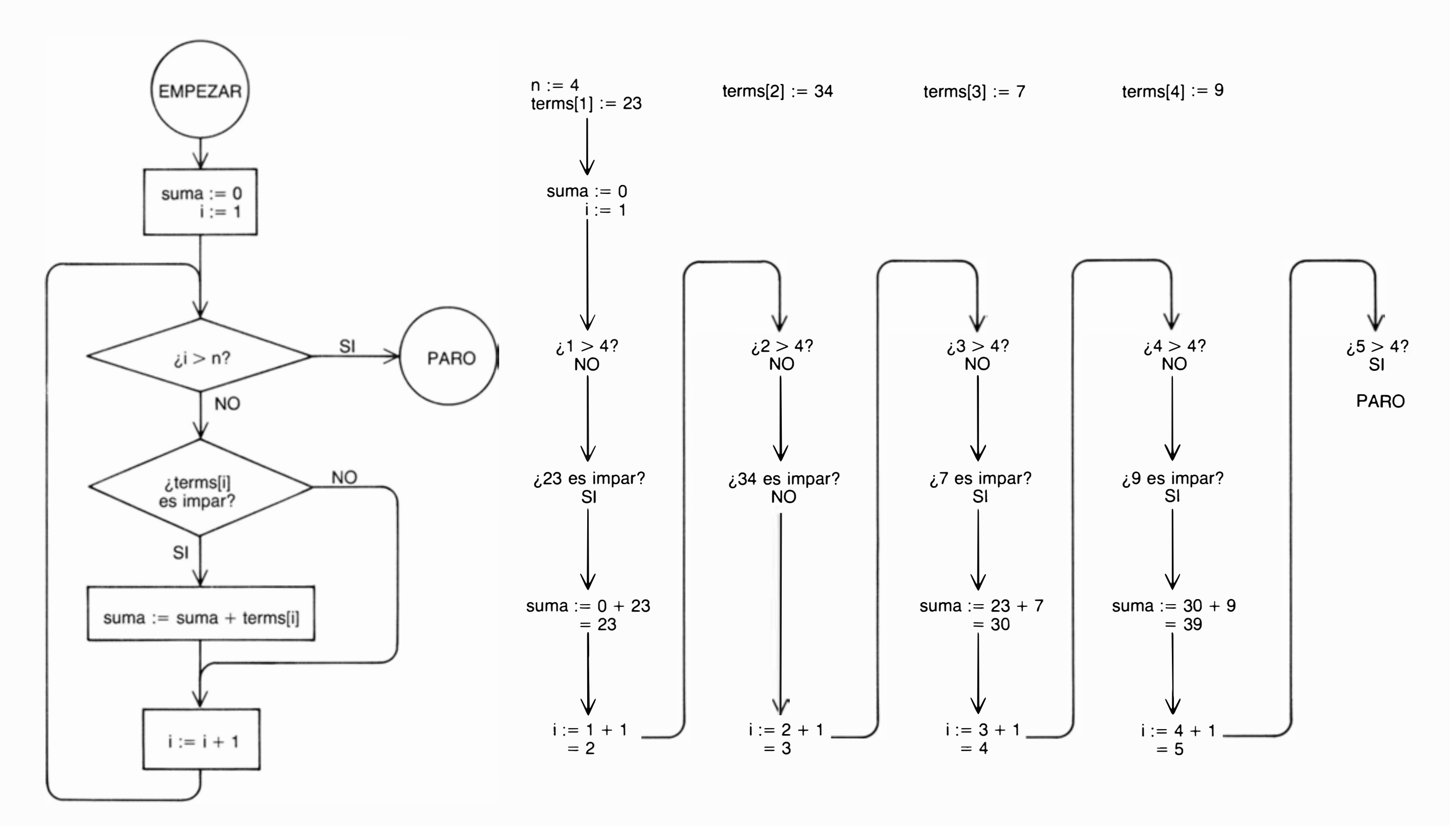


2. DESARROLLO DE UN PROGRAMA EN LOGO esquematizado en cinco momentos. El programa da instrucciones a una "tortuga", un dispositivo mecánico de dibujo. En la primera versión del programa, se explicitan todas las instrucciones necesarias para dibujar una estrella de cinco puntas. La orden *repeat* (repetir) de la segunda versión condensa el programa y reduce la probabilidad de ocurrencia de error. La tercera versión tiene la misma estructura básica de programa, pero dibuja una estrella mayor, de nueve puntas. En la cuarta versión se define un procedimiento donde la longitud del lado y el número de puntas son magnitudes variables. En la versión final del programa, una cláusula *if* (si) permite que el procedimiento se ejecute sólo si el número de puntas especificado es mayor que dos. Las órdenes *repeat* e *if* son ejemplos de estructuras de control importantes en la mayoría de los lenguajes de programación.

Entre los cientos o miles de lenguajes de programación apenas una docena gozan de un uso generalizado. Las figuras 4, 5, 6, 7, 8 y 9 ilustran un mismo problema programado en seis lenguajes distintos: BASIC, Pascal, COBOL, Forth, APL y Lisp. Se escogieron, en parte, por tratarse de lenguajes bien establecidos, con una población respetable de programadores que dominan su uso. Ponen, asimismo, sobre el tapete la ingente multiplicidad de formas en que una idea sencilla puede expresarse. En todos los casos se ha buscado un estilo natural o cómodo para el programador acostumbrado al lenguaje.

El problema no destaca por ningún interés intrínseco especial. Se escogió porque podía programarse rápidamente una solución en todos los lenguajes y porque demostraba los mecanismos esenciales requeridos para definir variables y procedimientos, así como para controlar la ejecución iterativa y condicional. El problema consiste en sumar los números impares de un conjunto de enteros.

El lenguaje de programación BASIC fue desarrollado en 1965 por John G. Kemeny y Thomas E. Kurtz, del Dartmouth College; pretendían, sobre todo, diseñar un lenguaje para cursos de introducción en informática y ciencias de la computación. De entonces acá ha decaído un tanto su fervor en el mundo académico, pero ha ganado popularidad en otros contextos, en particular dentro del ámbito de la programación de microordenadores. En BASIC, cada línea se identifica con un número; el control del flujo a lo largo del programa se funda, sobre todo, en remisiones al número de línea. El corazón del programa muestra es un bucle donde todos los mandatos incluidos entre un *FOR* y un *NEXT* (siguiente)



3. EJEMPLO DE CALCULO que nos ilustra las características de distintos lenguajes de programación. Consiste en hallar la suma de los números impares de un conjunto dado de *n* enteros. El algoritmo, con su organigrama a la izquierda, se materializa en los programas en Pascal, BASIC o COBOL que aparecen en las figuras 4, 5 y 6. El núcleo del programa lo constituye un bucle ejecutado *n* veces. En cada pasada a través del bucle se examina un término de la matriz; si es un número impar, se añade al total obtenido hasta el momento. A la derecha se resiguen los sucesivos valores tomados por las variables del procedimiento, según se van efectuando los cálculos para un vector de cuatro números. El símbolo ":=" confiere, a la variable a su izquierda, el valor computado a su derecha. Un número entre corchetes, como en *terms* [1], es equivalente a un subíndice: identifica un elemento del vector *terms*.

son ejecutados repetidamente. El cálculo real se hace en una orden de asignación, que empieza con la palabra clave *LET* (sea) y que confiere un nuevo valor a una variable.

El Pascal nacía en torno a 1970, diseñado por Niklaus Wirth, del Instituto Politécnico Federal de Zurich. Pensado también en su origen para la enseñanza, ha sido adaptado a otros muchos propósitos. A diferencia del BASIC, el Pascal requiere que el programador declare cada variable y especifique su tipo; en este caso, las variables son enteros y matrices de enteros. La remisión a procedimientos y funciones es por nombre, no ya por número de línea; ello mejora la legibilidad de los programas.

El Pascal goza en su haber de un mérito especial: ha sido buen progenitor de posteriores lenguajes. En este sentido, Wirth acaba de diseñar el Modula-2, lenguaje levantado sobre muchos de los conceptos introducidos en Pascal, si bien pone énfasis en la construcción del programa entendido como conjunto de módulos independientes. Ada, un lenguaje desarrollado bajo los auspicios del Departamento de Defensa norteamericano, descansa asimismo en el Pascal, aunque es más complejo.

En 1960, un comité conjunto de fabricantes de computadores y de usuarios creaba el COBOL, acróstico de Common Business-oriented Language. Durante largo tiempo ha sido el COBOL el principal lenguaje en proceso de datos a gran escala en las tareas de gobierno, banca, seguros y áreas similares. Un programa COBOL está formado por cuatro divisiones o partes: identificación, entorno, datos y procedimiento. La figura 6 muestra solamente la división de datos, donde se declaran las variables, y la división de procedimiento, donde se expresan los algoritmos. Muchos lenguajes de programación se modelan con la notación matemática o de la lógica formal; no así el COBOL, que se modela con la sintaxis de la oración inglesa. Los programas, altamente legibles, pecan, a menudo, de farragosos.

El lenguaje Forth lo inventó alrededor de 1970 Charles H. Moore, que trabajaba entonces en el National Radio Astronomy Observatory estadounidense. Pretendíase con él un lenguaje para el control de procesos, en particular, el control de telescopios; nuevamente, sin embargo, se difundió a otros dominios. Se ha adaptado a muchas aplicaciones en miniordenadores y en microordenadores, y ello en parte debido a que los programas Forth tienden a ocupar poco espacio en la memoria. A diferencia del COBOL, los programas Forth son de difícil lectura y tersos hasta el extremo; varias palabras clave son meros signos de puntuación.

En Forth la instalación central de la computadora es la "pila", un área de memoria organizada a modo de pila de bandejas en un autoservicio: la primera que se coloca en el montón es la última que se usará. En la versión en el lenguaje Forth del programa tomado como ejemplo se supone que la ristra de números y el tamaño de la misma están en lo alto de la pila cuando se llama a la función; todos los cálculos se hacen en la pila y el valor de la función se retorna a lo alto de la pila. No se han definido variables.

APL tiene una sintaxis más concisa que la del propio Forth. Aunque el nombre lo forman las iniciales de *A Programming Language* (un lenguaje de programación), cuando en 1961 se publicó el primer libro sobre APL (por Kenneth E. Iverson, de la compañía IBM), el lenguaje era una mera notación para expresar problemas en matemática aplicada; su acomodación al ordenador llegó después. Una caracterís-

```
program SumOddNumbers;
type TermIndex = 1..100;
     TermArray = array [TermIndex] of integer;
var  myTerms: TermArray;
function SumOdds(n: TermIndex; terms: TermArray): integer;
  var i: TermIndex;
      sum: integer;
  begin
    sum := 0;
    for i := 1 to n do
      if Odd(terms[i]) then
        sum := sum + terms[i];
    SumOdds := sum;
  end;
begin
  myTerms[1] := 23; myTerms[2] := 34; myTerms[3] := 7; myTerms[4] := 9;
  WriteLn(SumOdds(4, myTerms))
end.
```

4. PROGRAMA EN PASCAL para sumar los números impares de una matriz. Emplea una función llamada *SumOdds* (suma de impares) con dos parámetros: un entero *n* y un vector *terms*. La función consiste en las sentencias contenidas en el rectángulo coloreado; el resto del programa prepara un vector particular con el que *SumOdds* opera. En Pascal, cualquier variable debe introducirse en una declaración que da el tipo de la variable. Algunos tipos, como *integer* (entero), están incorporados en el lenguaje de programación; otros, como *TermIndex*, los define el programador. El bucle se designa con las órdenes *for... to... do...* (para... hacia... hacer...) y el condicional por *if... then...* (si..., entonces...).

```
 100 DIM T(100)
 200 READ N
 300 FOR I = 1 TO N
 400    READ T(I)
 500 NEXT I
 600 GOSUB 1100
 700 PRINT S
 800 GOTO 2000
 900 DATA 4
1000 DATA 23, 34, 7, 9
1100 REM MAKE S THE SUM OF THE ODD ELEMENTS IN ARRAY T(1..N)
1200 LET S = 0
1300 FOR I = 1 TO N
1400    IF NOT ODD(T(I)) THEN GOTO 1600
1500    LET S = S + T(I)
1600 NEXT I
1700 RETURN
2000 END
```

5. EN EL PROGRAMA EN BASIC se emplea una subrutina para sumar los términos impares de una matriz. La subrutina, indicada aquí por el recuadro en color, no tiene nombre y debe referenciarse con un número de línea; se le llama con la sentencia *GOSUB 1100*. Tampoco la subrutina BASIC tiene parámetros; los valores se asignan a variables "globales" a las que la subrutina puede acceder. Una variable no tiene que declararse en BASIC a menos que tenga subíndices, como en una matriz; en este ejemplo la declaración *DIM* (por dimensión) establece que el vector *T* puede tener hasta 100 elementos. La orden *FOR... NEXT...* define un bucle; a su vez, la instrucción *IF..., THEN...* define un condicional.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 NUMERIC-VARIABLES USAGE IS COMPUTATIONAL.
    02 TERMS PICTURE 9999 OCCURS 100 TIMES INDEXED BY I.
    02 N PICTURE 999.
    02 SUM PICTURE 999999.
    02 HALF-TERM PICTURE 9999.
    02 RMDR PICTURE 9.
PROCEDURE DIVISION.
EXAMPLE.
    MOVE 23 TO TERMS(1).
    MOVE 34 TO TERMS(2).
    MOVE  7 TO TERMS(3).
    MOVE  9 TO TERMS(4).
    MOVE  4 TO N.
    PERFORM SUM-ODDS.
SUM-ODDS.
    MOVE 0 TO SUM.
    PERFORM CONSIDER-ONE-TERM VARYING I FROM 1 BY 1 UNTIL I > N.
CONSIDER-ONE-TERM.
    DIVIDE 2 INTO TERMS(I) GIVING HALF-TERM REMAINDER RMDR.
    IF RMDR IS EQUAL TO 1; ADD TERMS(I) TO SUM.
```

6. PROGRAMA EN COBOL para el cálculo de la suma de los números impares. Usa el procedimiento *SUM-ODDS*, que recurre a otro procedimiento llamado *CONSIDER-ONE-TERM*. Un procedimiento COBOL no puede tener parámetros; por tanto, antes de llamar a *SUM-ODDS* con una orden *PERFORM*, se asignan valores a *N* y a los *N* primeros elementos de *TERMS*. Las palabras clave *PERFORM... VARYING...* definen el bucle e *IF...* introduce la cláusula condicional. En la división de datos los números 01 y 02 designan dos niveles en una jerarquía de estructuras de datos. *PICTURE* (imagen) especifica con qué formato deben aparecer los valores. Sólo se muestra un fragmento del programa completo.

tica que distingue al APL es que puede tratar con la misma facilidad un vector de números completo que un simple valor; un comando que suma dos números puede aplicarse sin cambios para sumar vectores con miles de elementos. El programa APL del ejemplo suma los elementos impares de un vector en un único mandato. Tomando el resto módulo 2 identifica los elementos impares, que son entonces extraídos del vector y sumados.

En determinados aspectos Lisp constituye el más simple de los lenguajes considerados aquí. Tiene una sola clase de mandato, una función; su potencia reside en que el valor devuelto por una función puede ser otra función. Lisp fue desarrollado a finales de los años 50 por John McCarthy, por aquel entonces en el Instituto de Tecnología de Massachusetts. Desde entonces se ha convertido en el lenguaje preferido de cuantos persiguen la meta de la inteligencia artificial. El nombre deriva de "list processing", pues tanto los programas como los datos se estructuran a modo de listas.

¿Qué haría el lenguaje Lisp con el programa ejemplo? Podría escribirlo como un bucle iterativo. Pero un programador en Lisp se sentiría más inclinado a escoger una técnica recursiva, en la cual un procedimiento se llama a sí mismo; el procedimiento llamado emite a su vez otra llamada a sí mismo, y así sucesivamente. Habrá que proporcionar algún medio de escape o la recursión se convertirá en una regresión infinita. El medio usual es una sentencia condicional dentro del procedimiento: cuando se satisface la condición, la ejecución retorna al inmediato nivel superior.

En el programa ejemplo, el conjunto de números toma la forma de una lista encadenada. Si la lista está vacía, la función devuelve un cero como resultado. Si el primer elemento de la lista es impar, se añade a la suma; la función se llama entonces a sí misma con un argumento que consiste en la lista que queda después de quitar el primer elemento. Llegará un punto en que cuando todos los elementos de la lista hayan sido extraídos, la cadena de cálculos pendientes habrá terminado.

Los ordenadores no se construyen para que "entiendan" el Logo, el BASIC u otros lenguajes que operen en un nivel de abstracción similar. La circuitería de una computadora reconoce sólo la materialización electrónica de los números binarios. Un programa almacenado de forma tal que pueda eje-

cutarse –esta forma se denomina código de máquina– es una secuencia de números binarios. Algunos de ellos representan instrucciones dadas al procesador central; otros son datos y unos terceros constituyen direcciones de memoria.

Nada impide escribir un programa directamente en código de máquina, pero resulta tedioso, con el agravante de que la probabilidad de completar sin error incluso un proyecto menor es reducida. (El ordenador no tiene problemas en distinguir 01101001 de 01011001 y en recordar lo que cada código significa, pero el ojo y la mente humanos no suelen aventajarle en tales tareas.) Hacia el final de la década de los 40 y principios de los 50, en un esfuerzo por relajar la fatiga de escribir código de máquina, los programadores inventaron una notación llamada código ensamblador. En vez de escribir los dígitos binarios para cada instrucción de máquina, el programador escribía una palabra corta o abreviatura, como ADD (agrega), SUB (sustrae) o MOVE (cambia). De un modo similar, la dirección en memoria donde se almacena una variable se reemplazó por un nombre asignado a la variable. Los valores numéricos se expresaron en notación decimal. Las palabras que representaban instrucciones se escogieron de suerte tal que fueran más fáciles de recordar que los valores binarios; así nacieron los nemotécnicos.

Al principio, la traducción de código ensamblador a código de máquina se hacía a mano. Es un proceso sencillo: una tabla registra la correspondencia fija entre los nemotécnicos de las instrucciones y sus códigos binarios; otra tabla similar puede construirse para los nombres de variables que aparezcan en un programa. El proceso es susceptible de mecanización, y no tardaron en aparecer los programas llamados ensambladores para realizar esta traducción.

Todavía se sigue haciendo alguna programación en el código ensamblador, pues ofrece acceso directo a todas las partes del ordenador. El código ensamblador que se haya escrito cuidadosamente es rápido y alcanza un buen rendimiento; y si hay que condescender entre la velocidad de ejecución y el tamaño del programa, el programador ejerce el control directo de las decisiones a tomar. Los ensambladores modernos son refinados programas traductores. Pero persiste todavía una aproximada correspondencia biunívoca entre las líneas de código ensamblador y las instrucciones de máquina, así que los programas tienden a ser bastante largos y las posibilidades de error no tienen techo. Las estructuras de control disponibles en la mayoría de los códigos ensambladores son primitivas. Y lo que es más importante, el código ensamblador se halla más cercano al lenguaje del ordenador que al del programador. Hay que expresar los algoritmos ajustados a lo que la máquina debe hacer, en vez de servirse de otros términos que podrían ser más naturales para una resolución del problema en cuestión.

A la hora de planificar la resolución

```
: SUMODDS
 0 SWAP 0
  DO
    SWAP DUP 2 MOD
      IF +
      ELSE DROP
      THEN
   LOOP
;
```

23 34 7 9 4 SUMODDS .

PALABRA	PILA							COMENTARIO
23	23							
34	23	34						
7	23	34	7					
9	23	34	7	9				
4	23	34	7	9	4			
SUMODDS	23	34	7	9	4			Llamar a SUMODDS.
0	23	34	7	9	4	0		
SWAP	23	34	7	9	0	4		
0	23	34	7	9	0	4	0	
DO	23	34	7	9	0			Quitar los valores de control del bucle.
SWAP	23	34	7	0	9			
DUP	23	34	7	0	9	9		
2	23	34	7	0	9	9	2	
MOD	23	34	7	0	9	1		
IF	23	34	7	0	9			CIM = 1; hacer desde IF hasta ELSE.
+	23	34	7	9				
ELSE	23	34	7	9				Saltar hasta THEN.
DROP	23	34	7	9				Saltado.
THEN	23	34	7	9				Saltado.
LOOP	23	34	7	9				Volver al DO.
DO	23	34	7	9				
SWAP	23	34	9	7				
DUP	23	34	9	7	7			
2	23	34	9	7	7	2		
MOD	23	34	9	7	1			
IF	23	34	9	7				CIM = 1; hacer desde IF hasta ELSE.
+	23	34	16					
ELSE	23	34	16					Saltar hasta THEN.
DROP	23	34	16					Saltado.
THEN	23	34	16					Saltado.
LOOP	23	34	16					Volver al DO.
DO	23	34	16					
SWAP	23	16	34					
DUP	23	16	34	34				
2	23	16	34	34	2			
MOD	23	16	34	0				
IF	23	16	34					CIM = 0; hacer desde ELSE hasta THEN.
+	23	16	34					Saltado.
ELSE	23	16	34					
DROP	23	16						
THEN	23	16						
LOOP	23	16						Volver al DO.
DO	23	16						
SWAP	16	23						
DUP	16	23	23					
2	16	23	23	2				
MOD	16	23	1					
IF	16	23						CIM = 1; hacer desde IF hasta ELSE.
+	39							
ELSE	39							Saltar hasta THEN.
DROP	39							Saltado.
THEN	39							Saltado.
LOOP	39							Ninguna iteración más.
;	39							Volver de SUMODDS.
.	< pila vacía >							Imprimir al resultado.

7. PROGRAMA FORTH para la suma de los números impares. No declara variables ni otras estructuras de datos, sino que trabaja exclusivamente con valores en una "pila". Cuando se llama a *SUMODDS*, se supone que los elementos a sumar están en la pila con el número de ellos encima. La línea que sigue a la definición del procedimiento debe teclearse para ejecutar el programa con una ristra de cuatro elementos. Se da la traza completa de la ejecución del programa, mostrándose el contenido de la pila tras la ejecución de cada palabra. Una "palabra" numérica como 0 o 2 empuja el número sobre la pila; *SWAP* (canje) intercambia los dos elementos de lo alto de la pila; *DUP* (por duplicar) empuja en la pila una copia del elemento situado en lo alto de la pila; *DROP* (soltar) elimina el elemento en lo alto de la pila. Los operadores como "+" o *MOD* (por módulo) sustituyen los dos elementos de lo alto de la pila por el resultado de la operación. La construcción del bucle, con *DO* (hacer), elimina dos elementos de la pila (digamos *i* y *j*) y ejecuta las palabras que siguen hasta *LOOP* (bucle) un total de *i-j* veces. El condicional *IF* (si) ejecuta las palabras entre *IF* y *ELSE* (si no) cuando el elemento en la cima de la pila (CIM) es distinto del cero; en caso contrario, ejecuta las palabras situadas entre *ELSE* y *THEN*. (Ilustración de Alan D. Iselin.)

de un problema, no nos atrae pensar en términos de registros y de direcciones de memoria; antes bien, es el propio problema el que sugiere la notación adecuada. Si se trata de un problema de física, el diseño del programa podría empezar con una ecuación del estilo $F = m\,a$; si de un asunto empresarial, la fórmula escogida podría ser *beneficios = ingresos-gastos*. Las operaciones especificadas por la fórmula se habrán de traducir ahora en instrucciones explícitas para la máquina. Los programadores del primer momento reconocieron que esta traducción era también susceptible de mecanizarse. Idea que subyace a la génesis del FORTRAN, BASIC y Pascal, entre otros. Durante algunos años los lenguajes de este tipo se llamaron lenguajes de alto nivel; parece más apropiado ahora referirse a ellos simplemente como lenguajes de programación, debido a que el código de máquina y el código ensamblador realmente no se califican como lenguajes.

Desde 1960 se han escrito muchos soportes logicales con la ayuda de lenguajes de programación, dotados estos últimos de muchas ventajas sobre representaciones a más bajo nivel. Puesto que una orden puede originar muchas instrucciones de máquina, los programas tienden a ser más cortos; ello reduce el trabajo invertido en su redacción y mejora además su claridad. Trabajar con conceptos pertinentes al problema en vez de atenerse a los definidos por la máquina rebaja, asimismo, la probabilidad de error. Más aún, introduce la posibilidad de "independencia de la máquina": escritura de un único programa que pueda ser ejecutado en muchos ordenadores.

Importa distinguir entre el lenguaje de programación y la realización ("implementation") del lenguaje. En sí mismo, el lenguaje es la notación, el conjunto de reglas que definen la sintaxis de un programa válido. Por realización del lenguaje se entiende el programa que convierte la notación de alto nivel en secuencias de instrucciones de máquina.

La realización de lenguajes ocurre a través de dos vías principales: compiladores e interpretadores. El compilador traduce todo el texto de un programa de alto nivel en un proceso continuo, creando un programa en código de máquina completo que puede entonces ejecutarse con independencia del compilador. Trabajar en un lenguaje compilado exige, por lo común, tres etapas: crear primero el texto del programa con un editor de textos o con un programa procesador de palabras; compilar, a continuación, el texto y, finalmente, ejecutar el programa compilado. El término compilador lo acuñó en 1951 Grace Murray Hopper, a la sazón en Remington-Rand Univac, para describir su primer programa traductor. Como una parte del proceso de traducción, el programa recobraba secuencias estándar de instrucciones de máquina a partir de unas tablas almacenadas en cinta magnética y las compilaba en un programa completo.

El intérprete ejecuta un programa a razón de una orden por vez, transformando cada construcción de alto nivel en instrucciones de máquina. La diferencia entre un compilador y un intérprete es parecida a la que media entre un traductor de obras literarias y un intérprete oral. El traductor literario toma un manuscrito completo y entrega un nuevo texto en otro lenguaje. El intérprete oral proporciona la traducción de cada frase u oración al hilo del discurso. Cierto es que muchos programas intérpretes acometen algún tipo de proceso inicial del texto antes de que empiecen la ejecución; las palabras clave se convierten en símbolos más cortos y los nombres de las variables se remplazan por direcciones. Aun con todo, una y otra clase de realización del lenguaje continúan siendo distintas: para que un programa interpretado pueda ejecutarse, el intérprete debe persistir en la memoria principal, mientras que, una vez compilado un programa, la presencia del compilador ya no es necesaria.

En principio, cualquier lenguaje de programación podría ser interpretado o compilado, indistintamente; pero en la mayoría de los casos la costumbre se ha inclinado por una opción determinada. FORTRAN, COBOL y Pascal suelen compilarse; Logo, Forth y APL casi siempre se interpretan; BASIC y Lisp se encuentran fácilmente en ambas formas. La ventaja capital de la compilación es la velocidad. El intérprete debe determinar una secuencia apropiada de instrucciones cada vez que se ejecuta un mandato; por tanto, resulta casi inevitable que un lenguaje interpretado sea más lento.

```
    ∇ SUM ← SUMODDS TERMS
[1]   SUM ← + /(2 ∣ TERMS) /TERMS
    ∇

SUMODDS 23 34 7 9

             TERMS ← 23   34   7   9     Asignación del valor inicial.
         (2 ∣ TERMS) ←  1    0   1   1     Vector de restos.
   (2 ∣ TERMS) /TERMS ← 23        7   9     Compresión de dos vectores.
+ /(2 ∣ TERMS) /TERMS ← 23   +    7 + 9     Reducción por suma.
               SUM ← 39                    Asignación del resultado.
```

8. PROGRAMA APL, que calcula la suma de los elementos impares de una matriz con una función cuya operación se especifica en una sola línea. La función tiene un parámetro, *TERMS*, una matriz que "sabe" cuántos elementos tiene; así pues *N* no necesita aparecer en el programa. Una sentencia APL se ejecuta de derecha a izquierda, excepto donde los paréntesis alteren el orden de evaluación. En este ejemplo, la expresión (*2* | *TERMS*) se evalúa primero; calcula los restos de dividir cada elemento de *TERMS* por 2 y crea una matriz del mismo tamaño que *TERMS* para contenerlos. El símbolo "/" puede denotar dos operaciones diferentes; ambas aparecen en el ejemplo. En la expresión (*2* | *TERMS*)/*TERMS*, "/" es una operación de "compresión" que crea una nueva matriz en la cual cada elemento de *TERMS* aparece sólo sí el correspondiente elemento de (*2*|*TERMS*) no es cero. En el símbolo "+/", "/" es un operador de "reducción", pues reduce la matriz a un único número, insertando un "+" entre cada par de elementos.

```
(DEFUN SUMODDS
  (LAMBDA (TERMS)
    (COND
      ((NULL TERMS) 0)
      ((ODD (CAR TERMS)) (PLUS (CAR TERMS) (SUMODDS (CDR TERMS))))
      (T (SUMODDS (CDR TERMS))))))

(SUMODDS '(23 34 7 9))

(SUMODDS '(23 34 7 9))
   = (PLUS 23 (SUMODDS '(34 7 9)))
     = (PLUS 23 (SUMODDS '(7 9)))
       = (PLUS 23 (PLUS 7 (SUMODDS '(9))))
         = (PLUS 23 (PLUS 7 (PLUS 9 (SUMODDS '( )))))
       = (PLUS 23 (PLUS 7 (PLUS 9 0)))
     = (PLUS 23 (PLUS 7 9))
   = (PLUS 23 16)
 = 39
```

9. PROGRAMA LISP. Este programa calcula la suma de los elementos impares por medio de una función que se llama a sí misma recursivamente. Una función Lisp es una lista, donde el primer elemento (llamado el *CAR*) es el nombre de la función y el resto de la lista (el *CDR*) da los parámetros. *DEFUN* es una función de definición de funciones; *LAMBDA* precede a los nombres de los parámetros; aquí el único parámetro es la lista de números *TERMS*. *COND* es una función condicional que evalúa el *CAR* de la lista que forman sus parámetros. Si el resultado es cierto, o *T* (de "true"), se evalúa el *CDR* de la lista; de otro modo *COND* continúa con la siguiente lista. Aquí hay tres posibilidades. Si *TERMS* es una lista vacía, *NULL* es verdad y *SUMODDS* devuelve un valor nulo. Si el *CAR* de *TERMS* es impar, se añade el *CAR* al total hasta el momento y se llama a *SUMODDS* para evaluar el *CDR* de *TERMS*. Si ninguna de estas condiciones es cierta, se alcanza la cláusula *T* (forzosamente cierta); en ella simplemente se llama a *SUMODDS* con (*CDR(TERMS)*) como su parámetro. En cada llamada se dejan los cálculos pendientes.

	CODIGO DE ENSAMBLAJE		
CODIGO DE MAQUINA	ETIQUETAS	INSTRUCCIONES	COMENTARIOS
00100100 01011111	SUMODDS	**MOVE.L** (A7)+ ,A2	Lleva a A2 la dir. de retorno de la pila.
00100010 01011111		**MOVE.L** (A7)+ ,A1	Lleva a A1 la dirección del primer término.
00110010 00011111		**MOVE.W** (A7)+ D1	Lleva n a D1.
01000010 01000010		**CLR.W** D2	Asigna un valor 0 a la suma en D2.
01001110 11111010 00000000 00001110		**JMP** COUNT	Salta al final del bucle y ve si n = 0.
00001000 00101001 00000000 00000000 00000000 00000001	BUCLE	**BTST** 0,1(A1)	Si el término señalado por A1 es par...
01100111 00000010		**BEQ.S** NEXT	... entonces ir a SIGUIENTE.
11010100 01010001		**ADD.W** (A1),D2	... si no, añadir el térm. a suma en D2.
01010100 01001001	SIGUIENTE	**ADDQ.W** #2,A1	Pon en A1 la dir. del siguiente térm.
01010001 11001001 11111111 11110010	TOTAL	**DBF** D1,LOOP	Disminuir D1; ir a BUCLE, salvo si es − 1.
00111110 10000010		**MOVE.W** D2, − (A7)	Trasladar la suma desde D2 a la pila.
01001110 11010010		**JMP** (A2)	Ir a la dirección de retorno.

10. EL CODIGO DE MAQUINA y el código de ensamblaje especifican los pasos del cálculo con los elementos impares en términos de los recursos físicos (hardware) del ordenador. El código es necesariamente específico de cada máquina concreta; en este caso, se trata del microprocesador 68000 de Motorola. El algoritmo empleado es muy parecido al del procedimiento *SumOdds* en Pascal, aunque es más compacto que el código que se generaría por un compilador de Pascal. Los parámetros se pasan al procedimiento en una pila; el resultado se devuelve a la pila. La dirección en que se reanuda la ejecución cuando el procedimiento ha acabado se encuentra también en la pila. La versión en código de ensamblaje del programa (*derecha*), donde las instrucciones toman la forma de abreviaturas "nemotécnicas", puede ser traducida directamente en el código de máquina binario que el microprocesador ejecuta.

Por otro lado ocurre que un lenguaje interpretado le conviene, a menudo, al programador, pues se acomoda mejor a un estilo interactivo de desarrollo de programas. Las diversas secciones de un programa pueden escribirse, probarse y ejecutarse sin abandonar el intérprete, y, si se encuentra un error, éste puede corregirse inmediatamente sin necesidad de retornar a un programa editor de textos y después compilar el programa de nuevo.

El mecanismo interno de un compilador o de un intérprete es demasiado complejo para abordarlo con detalle. Sí podemos dar las líneas generales que definen la estructura de un compilador arquetípico. Hay al menos tres fases en el proceso de compilación. Consiste la primera en un análisis lexicográfico, donde el compilador identifica los diversos símbolos que aparecen en el texto del programa y los clasifica como palabras clave, valores numéricos, nombres de variables, etcétera. En la siguiente fase, de análisis sintáctico, el compilador determina las relaciones sintácticas de las palabras clave y construye una representación esquemática de la estructura del programa. Cada *if*, por ejemplo, se asocia con un subsiguiente *then* (si..., entonces...). En la tercera fase, se genera el código de máquina correspondiente a la estructura analizada. Algunos compiladores añaden una cuarta fase de optimización, en la cual se revisa el código para mejorar su rendimiento.

A lo largo de los últimos 30 años se ha venido prestando especial atención al diseño de compiladores, gracias a lo cual disponemos ya de una metodología bien desarrollada para su construcción. Hay que empezar por definir el propio lenguaje en una forma explícita. Se ha convertido en práctica común especificar la gramática en términos de "reglas de producción" que pueden aplicarse de una manera recursiva para generar todas las órdenes posibles del lenguaje. La creación del compilador se torna entonces un trabajo de programación bastante sencillo; hay incluso compiladores de compiladores, que automatizan parte de la tarea.

La idea de un lenguaje de programación flotaba en el ambiente desde que aparecieron los computadores digitales. En el año 1945, el matemático alemán Konrad Zuse inventó una notación a la que puso por nombre Plankalkül. Las órdenes del lenguaje tenían un formato bidimensional. Las variables y sus subíndices se alineaban verticalmente; las operaciones a realizar con ellas se disponían a lo largo del eje horizontal. Zuse escribió programas Plankalkül sobre el papel –uno de ellos simulaba movimientos de ajedrez–, pero no llegó a poner en práctica su lenguaje. Muchas de las ideas que desarrolló, sin embargo, han sido introducidas en los lenguajes modernos.

De todos los lenguajes de programación el de mayor influencia fue, sin duda, el FORTRAN, desarrollado por John Backus y sus colegas, en IBM, entre los años 1954 y 1957. El nombre contrae la expresión "formula translation", traductor de fórmulas. El lenguaje se inventó para cálculos científicos y numéricos, campo donde sigue en uso. Su aparición fue recibida con considerable escepticismo, pues las máquinas computadoras eran por aquel entonces un recurso escaso y valioso, lo que obligaba a insistir en la eficacia de los programas. Creíase que un lenguaje de más alto nivel comprometería inevitablemente el rendimiento. Backus y su equipo realizaron, empero, una extraordinaria hazaña: crearon un compilador cuyo producto igualaba, en calidad, al de un programa codificado a mano.

Por las mismas fechas, Hopper y sus colegas en Remington-Rand Univac desarrollaron un lenguaje de programación, llamado Flow-Matic, para el proceso de datos en los negocios. Aunque menos refinado que el FORTRAN, la experiencia ganada con él a lo largo de varios años de uso fue el trampolín para el COBOL. Otro lenguaje importante introducido a finales de los 50 fue el Algol (que deriva de "*algo*rithmic *l*anguage", lenguaje algorítmico). El Algol-58, la primera versión, fue diseñado por un comité internacional que aunó la sintaxis pragmática del FORTRAN con la notación, más elegante, del Plankalkül. Resultó así un lenguaje legible y práctico, desempeñando un papel importante en el advenimiento de lenguajes posteriores, incluido el Pascal.

Otra buena gavilla de lenguajes remontan sus raíces a la misma época. Para el análisis de textos se creó COMIT; APT, para el control de máquinas-herramienta. JOVIAL, derivado del Algol, fue el primer lenguaje de propósito múltiple que gozó de amplio uso. Se mostró idóneo para aplicaciones científicas y la gestión comercial. Al principio de los sesenta apareció el Lisp y también lo hizo la notación del APL, aunque no su realización.

La rápida proliferación de lenguajes inquietó a muchos observadores. Después de todo, la mayor parte del trabajo matemático se hace con una única notación universalmente aceptada. La realización de un nuevo lenguaje no es empeño menor, y sentirse cómodo programando en él lleva, asimismo, su tiempo. Pronto se ofrecieron varios proyectos para diseñar un nuevo lenguaje tan completo y versátil que pudiese servir de argot universal de la programación. Esfuerzos que, todos, acabaron en fracaso. El éxito parcial del PL/I, desarrollado bajo el patrocinio del IBM en 1965, demostró que un len-

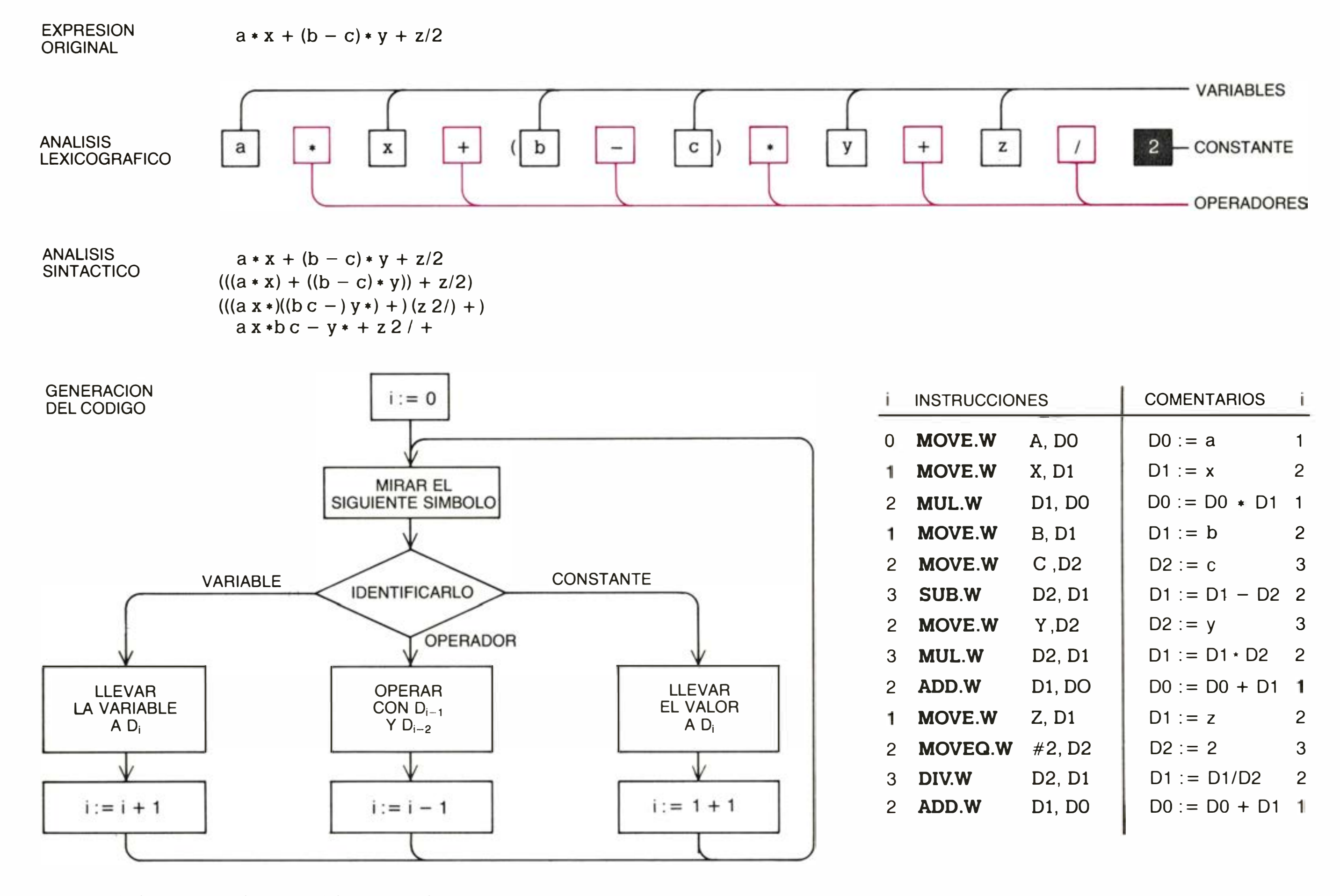
guaje multipropósito resultaba probablemente duro de aprender y difícil de realizar. Y a medida que las técnicas de computación se diversificaron, la gente se dio cuenta de que continuarían siendo necesarios nuevos lenguajes para afrontar áreas de aplicación especiales.

En cierto modo, toda la investigación en lenguajes de programación desarrollada desde 1957 ha venido instada por el deseo de corregir los defectos descubiertos en el FORTRAN. El propio FORTRAN se ha sometido varias veces a revisión. La versión original imponía ciertas restricciones arbitrarias al programador –por ejemplo, un nombre de variable no podía tener más de seis caracteres– y ofrecía solamente facilidades limitadas para la definición de estructuras de datos. Quizá las deficiencias más serias residieran en los dispositivos de control del flujo del programa. Todos los puntos de bifurcación tenían que definirse con números de línea y, si no se andaba con el cuidado necesario, la función de un programa podía quedar bastante oscura bajo una maraña de órdenes *GOTO*. En posteriores versiones se introdujeron estructuras de control que estimulaban un estilo de programación más legible.

Todos los lenguajes de los que me he venido ocupando hasta aquí pueden clasificarse como lenguajes de procedimiento o prescriptivos. El programa escrito en cualquiera de ellos establece cómo obtener un resultado; nos dice haz primero esto, luego haz aquello, y así sucesivamente. Pero existen otros que no son lenguajes de procedimiento: los descriptivos; cuya importancia crece por días. Los programas descriptivos establecen qué resultado se desea obtener sin especificar cómo obtenerlo. El programa traza relaciones, no el flujo de control, y así el programador se ve relevado de la responsabilidad de desarrollar los diversos pasos de un algoritmo y especificar su orden.

Los lenguajes descriptivos más destacados son los programas de hojas electrónicas, como el VisiCalc y el MultiPlan, que la proliferación de ordenadores personales ha popularizado. Para especificar un cálculo en MultiPlan, se escriben fórmulas, algo común con el BASIC o el FORTRAN. El orden en que las fórmulas se evaluarán, sin embargo, se determina por su realización, no por el programador. Hasta cierto punto, las relaciones temporales se substituyen por relaciones espaciales. En un lenguaje convencional, la salida de resultados de un procedimiento puede servir de entrada de datos para el siguiente; la noción análoga en una hoja electrónica determina que el valor de una celda dependa del valor de la otra.

En el lenguaje Prolog se advierte incluso menos tendencia a definir procedimientos. Prolog deriva del Lisp y últimamente ha atraído la atención de muchos estudiosos en el campo de la inteligencia artificial. En dicho lenguaje no se escriben fórmulas, sino que se definen relaciones entre objetos y cantidades. Consta sólo de declaraciones, sin



i	INSTRUCCIONES		COMENTARIOS	i
0	**MOVE.W**	A, D0	D0 := a	1
1	**MOVE.W**	X, D1	D1 := x	2
2	**MUL.W**	D1, D0	D0 := D0 * D1	1
1	**MOVE.W**	B, D1	D1 := b	2
2	**MOVE.W**	C ,D2	D2 := c	3
3	**SUB.W**	D2, D1	D1 := D1 − D2	2
2	**MOVE.W**	Y ,D2	D2 := y	3
3	**MUL.W**	D2, D1	D1 := D1 * D2	2
2	**ADD.W**	D1, DO	D0 := D0 + D1	1
1	**MOVE.W**	Z, D1	D1 := z	2
2	**MOVEQ.W**	#2, D2	D2 := 2	3
3	**DIV.W**	D2, D1	D1 := D1/D2	2
2	**ADD.W**	D1, D0	D0 := D0 + D1	1

11. FUNCIONAMIENTO DE UN COMPILADOR, o traductor, para un lenguaje de programación. Consta, por lo menos, de tres fases, que se muestran aquí para el caso sencillo de una expresión aritmética en Pascal. En el análisis lexicográfico, los símbolos que constituyen el programa se identifican y categorizan. El análisis sintáctico define las relaciones semánticas entre símbolos. En una expresión aritmética la principal tarea del analizador sintáctico consiste en determinar qué operandos están asociados con cada operador. Aquí se hace comparando la precedencia de los operadores adyacentes (supuesto que la multiplicación precede a la división, ésta a la suma, y así sucesivamente); se añaden paréntesis alrededor de las operaciones de una precedencia más alta. La expresión se convierte en notación "postfija" intercambiando el operador y el segundo operando en cada subexpresión. Se quitan entonces los paréntesis, resultando una expresión que puede evaluarse de izquierda a derecha. La generación de código transforma la expresión producto del analizador en instrucciones de máquina, recurriendo a un sencillo algoritmo de asignación de cada variable a un registro de soporte físico del ordenador.

órdenes. Así, la relación (*producto altura anchura área*) describe la igualdad *área = altura × anchura*, pero no especifica que la altura y la anchura sean las cantidades conocidas, o que el área sea la que va a calcularse. La misma relación puede servir para hallar la altura cuando se conocen el área y la anchura.

En la evolución de los lenguajes de programación se advierte otra tendencia: el interés creciente por los sistemas notacionales llamados lenguajes orientados hacia el objeto. Como ya se mencionó, las aseveraciones de la mayoría de los lenguajes de programación son imperativas: no se nombra la entidad a la que uno se está dirigiendo, por la simple razón de que hay una única posibilidad, la encarnación abstracta del ordenador como un todo. En un lenguaje orientado hacia el objeto, la computadora se divide conceptualmente en objetos a los que uno puede dirigirse por separado. Además, los objetos pueden comunicarse entre sí a través de mensajes.

La noción de objetos logicales fue introducida en Simula 67 por Ole-Johan Dahl y Kristen Nygaard, del Centro Noruego de Computación en Oslo. Simula 67 es un lenguaje derivado del Algol 60. La idea no despertó mayor interés hasta el desarrollo del lenguaje Smalltalk, en los años setenta, por Alan Kay y un grupo de colegas (entre los que me contaba) en el Centro de Investigación de Xerox en Palo Alto. El Smalltalk consta exclusivamente de construcciones orientadas al objeto, lo cual hace que la especificación del lenguaje sea pequeña y muy general; por otro lado, y debido a que cualquier cosa del lenguaje siempre es un objeto, de algunos mecanismos importantes de estructuración de datos no puede sacarse el rendimiento apetecido.

Todo objeto logical consta de estructuras de datos y algoritmos. Cada objeto "sabe" cómo operar con sus propios datos; mas, para el resto del programa, el objeto viene a ser una caja negra, cuyo funcionamiento interno no importa. La verdad es que objetos distintos pueden emplear algoritmos diferentes para llevar a cabo tareas que el programador identifica con la misma palabra clave. Del mismo modo que pingüinos, caballos y ciempiés siguen métodos dispares para la actividad que genéricamente se identifica como andar, así también objetos cuyos datos consisten en enteros, matrices y números complejos emplearían métodos distintos para la operación de sumar.

Mis colegas y yo hemos desarrollado,

```
añadir (Adán es-uno-de-los-padres-de Caín)
añadir (Adán es-uno-de-los-padres-de Abel)

añadir (Eva es-uno-de-los-padres-de Caín)
añadir (Eva es-uno-de-los-padres-de Abel)
añadir (Caín es-uno-de-los-padres-de Enoch)

cuál (x : x es-uno-de-los-padres-de-Abel)
Adán
Eva
No (más) respuestas

cuál (x : Eva es-uno-de-los-padres-de x)
Caín
Abel
No (más) respuestas

añadir (x es-un-antepasado-de y si x es-uno-de-los-padres-de y)
añadir (x es-un-antepasado-de y si z es-uno-de-los-padres-de y
        y x es-un-antepasado-de z)

cuál (x : x es-un-antepasado-de Enoch)
Caín
Adán
Eva
No (más) respuestas

cuál (x : Adán es-un-antepasado-de x)
Caín
Abel
Enoch
No (más) respuestas
```

12. LENGUAJE DESCRIPTIVO. El llamado Prolog no tiene mandatos; consiste enteramente en declaraciones. En otras palabras, un programa Prolog no da instrucciones explícitas de cómo efectuar una operación; se limita a establecer relaciones y hacer inferencias a partir de las mismas. La ilustración muestra un programa en un dialecto llamado Micro-Prolog. Las cinco primeras declaraciones expresan ciertas relaciones paterno-filiales. El sistema puede entonces responder a preguntas sobre los hechos establecidos, por ejemplo identificar los padres de Abel y los hijos de Eva. Se introducen seguidamente dos reglas de inferencia para definir la relación "antepasado de" en términos de la relación "ser uno de los padres de". El sistema puede aplicar las reglas para hallar los antepasados o los descendientes.

en Apple Computer, Inc., un lenguaje al que hemos bautizado Clascal. Añade la noción de clases de objetos a la estructura subyacente de Pascal. Clascal, Smalltalk, Simula y algunos otros orientados hacia el objeto permiten que los objetos de una clase hereden propiedades de la superclase a la que pertenezcan; así pues, cada clase no tiene que construirse a partir de cero. Sólo se han de especificar los rasgos que distingan a una clase individual. Volviendo al ejemplo: pingüinos, caballos y ciempiés comparten la noción de patas, pero difieren en su número y en los pormenores del método de locomoción. Esta herencia constituye otro mecanismo de abstracción que permite que las propiedades de una clase las exploten muchas subclases.

La herencia resulta ser particularmente útil en el diseño de logicales para gráficos interactivos, otra parcela donde se trabaja activamente. Lenguajes de programación enteros pueden construirse con imágenes gráficas. Hasta ciertos juegos de ordenador que se apoyan en gráficos poseen características de lenguaje de programación. Nos sirve de valioso ejemplo Robot Odyssey I, juego lanzado recientemente por Learning Company; "robots" programados al conectar puertas lógicas electrónicas y otros componentes en una pantalla de vídeo incorporan los conceptos de ejecución condicional y de definición de procedimiento a seguir. Jaron Z. Lanier y sus colegas, del VPL Research de Palo Alto, tienen en fase de desarrollo un lenguaje de programación visual, completo, que se conoce provisionalmente como Mandala. Un ejemplo de lo que un programa Mandala podría parecer se muestra en la portada de este número.

Los lenguajes de programación conocen hoy otra dirección de expansión en la de la explotación del cómputo en paralelo en sistemas formados por procesadores múltiples. Diríase que 100 unidades de proceso deberían ser capaces de resolver un problema 100 veces más rápido que un único procesador de la misma velocidad intrínseca, pero esta ganancia podrá sólo conseguirse si el logical logra seccionar el problema en muchas piezas que puedan trabajarse simultáneamente.

Algunos lenguajes proporcionan un mecanismo explícito de designación de tareas que pueden ejecutarse en paralelo; tal ocurre con el Occam, desarrollado por Inmos, compañía de semiconductores. Otros lenguajes dejan que el compilador analice el programa y descubra oportunidades de ejecución en paralelo. Entre éstos citemos el COMPEL (abreviatura de cómputo paralelo) en el que colaboré con Horace J. Enea en 1969. Un programa COMPEL consta sólo de órdenes de asignación, las cuales no son ejecutadas necesariamente en la secuencia en que se escriben; se supone que el compilador deducirá qué instrucciones debe ejecutar primero. No se ha escrito ningún compilador para programas COMPEL, pero se han puesto en operación desde entonces otros lenguajes con un mecanismo similar (denominados lenguajes de flujo de datos).

La gran diversidad de lenguajes de programación hace imposible alinearlos de acuerdo con una escala única de medida. No existe el mejor lenguaje de programación, como tampoco el mejor lenguaje natural. "Yo hablo en español con Dios, en italiano con las mujeres, en francés con los hombres y en alemán con mi caballo", dijo Carlos V (seguramente en francés). También un lenguaje de programación debe escogerse de acuerdo con el fin que nos guíe.

Ciencia y sociedad

Cajal y el saber científico

Hay sabios que no sienten dentro de sí la comezón de preguntarse por la consistencia y el sentido de las verdades científicas a cuyo descubrimiento han consagrado su vida; con genialidad mayor o menor se limitan a ser *chiffonniers de faits*, "traperos de hechos", como de sí mismo dijo el fisiólogo Magendie. Hay otros, en cambio, que en un momento u otro de su vida, cuando por la razón que sea dejan el trabajo cotidiano y se quedan solos consigo mismos y con su obra, se sienten íntimamente movidos a preguntarse: más allá de su formulación inmediata, ¿qué verdad y qué realidad tienen los hechos que yo he descubierto? En mi vida personal y en la vida de los hombres todos, ¿qué sentido tienen mis descubrimientos y mi esfuerzo para lograrlos? De este linaje fueron, entre otros, Claudio Bernard, Planck y Einstein; de él fue también, y por modo eminente, nuestro Cajal.

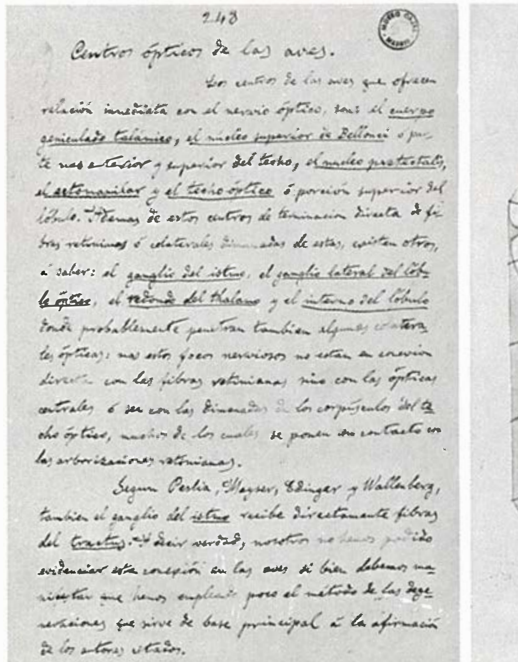
A lo largo de su fecundísima vida, Cajal fue un incesante descubridor de hechos científicos nuevos. Más que entre todo los restantes neurólogos juntos habría descubierto, según el fehaciente testimonio del sabio italiano Ernesto Lugaro. Hechos y leyes, como la que rige la transmisión intraneuronal del impulso nervioso. No pudo tener tiempo para más, se dice uno. Pero la mente de Cajal no era sólo de investigador, era también de pensador, de sabio, en el más plenario sentido de esta palabra; y así, cuantas veces la vida le obligó a interrumpir por unos días su trabajo histológico, desde dentro de sí mismo se sintió movido a pensar acerca de su propio saber científico y sobre el saber científico en general. Cinco fueron las más importantes de tales ocasiones: su discurso de ingreso en la Real Academia de Ciencias (1895), que más adelante daría lugar al libro *Reglas y consejos para la investigación científica*; su discurso en el homenaje que le tributó la Universidad de Madrid cuando le fue otorgado el Premio de Moscú (1900); la conferencia con que contribuyó a la celebración nacional del tercer centenario del *Quijote* (1905); la definitiva composición de *Recuerdos de mi vida* (1922); la redacción de *El mundo visto a los ochenta años* (1932).

Consideremos sinópticamente esos cinco testimonios de su pensamiento y tratemos de reducir a sistema lo que concierne a nuestra reflexión ahora: qué fue el saber científico para nuestro gran sabio. Lo cual nos obliga a discernir dos cuestiones sucesivas: la adquisición del saber científico por el sabio que lo conquista y la significación de ese saber, tanto para el sabio mismo como para quienes de él lo reciben.

I. *Asombro e interrogación ante la realidad.* La iniciación técnica de un trabajo científico –por ejemplo: la aplicación del método de tinción de Golgi al estudio de la textura del cerebelo, uno de los muchos que Cajal emprendió– tiene como inmediato presupuesto el ejercicio de dos actividades de la mente, el asombro y la interrogación.

Vieja noción, la fundamental importancia del asombro para el descubrimiento de nuevas verdades. "Principio de la filosofía", le llama Platón. Sólo "quien se asombra y duda se halla en el buen camino hacia la sabiduría", dirá Aristóteles. Tesis ambas que hoy deben ampliarse a la adquisición original de todo saber, sea éste científico o filosófico.

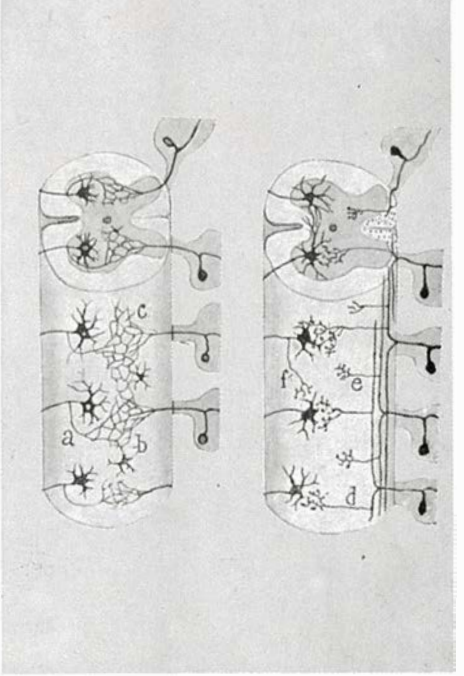
Como para demostrar con su propia experiencia la verdad de tan venerable enseñanza, Cajal nos hace ver que toda su vida intelectual, desde su infancia en la aldea aragonesa de Valpalmas, fue una ascendente sucesión de asombros precientíficos y científicos. Al niño de Valpalmas y Ayerbe le asombran –le admiran, dice textualmente– "los esplendores del sol, la magia de los crepúsculos, las alternativas de la vida vegetal, con sus fastuosas fiestas primaverales..."; todos los aspectos, triviales para los más, del maravilloso rostro de la naturaleza. El rayo que cayó en la escuela de Valpalmas le llenó de estupor, porque le hizo advertir en el mundo cósmico, hasta entonces contemplado como "perpetuo milagro", la existencia de "una fuerza ciega e incontrastable, indiferente a la sensibilidad". Un eclipse de sol fue para él "luminosa revelación", porque se produjo exactamente cuando los astrónomos lo habían anunciado. Poco más tarde, ya en Ayerbe, su frecuente reclusión punitiva en un estrecho recinto semisubterráneo le permite descubrir en el techo, tras el asombro de percibirlas, las imágenes invertidas de la cámara oscura. "¡Cuántos hechos interesantes! –dirá más tarde, recordando la alegre indiferencia con que sus compañeros de encierro acogieron la noticia de su hallazgo– dejaron de convertirse en descubrimientos fecundos, por haber creído sus


248

Centros ópticos de las aves.

Los centros de las aves que ofrecen relación inmediata con el nervio óptico, son: el cuerpo geniculado talámico, ...

Segun Perlia, Mayser, Edinger y Wallenberg, tambien el ganglio del istmo recibe directamente fibras del tractus.



1. *Página del manuscrito de su estudio monográfico* **Estructura de los centros nerviosos de las aves**, *publicado en 1888. La ilustración de la derecha es un esquema de Cajal destinado a comparar la concepción de Golgi acerca de las comunicaciones sensitivo-motrices de la célula espinal (I) con sus investigaciones (II).*

primeros observadores que eran *cosas naturales y corrientes*, indignas de análisis y meditación! ¡Oh, la nefasta inercia mental, la *inadmirabilidad* de los ignorantes!"

La contemplación ingenua del artificio técnico y el precoz descubrimiento, en los primeros libros escolares, de lo que la ciencia –la penetración de nuestra inteligencia en la realidad de las cosas– es para la mente del hombre, son el segundo paso en la carrera de asombros que en medio de sus tan conocidas travesuras es la vida infantil de Cajal. La deflagración de la pólvora le llena de indefinible sorpresa: "cada estallido de un cohete, cada disparo de arma de fuego eran para mí estupendos milagros". El ferrocarril, novísimo entonces en España, "fue el primero de mis asombros". El revelado de la placa fotográfica, visto por él en la elemental cámara oscura de un fotógrafo ambulante, "causóme –dice– indecible asombro y hasta verdadera estupefacción".

Los estudios médicos pusieron ante los ojos de Cajal una maravilla nueva, el cuerpo humano. El cadáver dejó pronto de ser objeto repulsivo para hacerse deleitoso campo de sorpresas: "Ante la imponente losa anatómica, protestaron al principio cerebro y estómago; pronto vino, empero, la adaptación. En adelante vi en el cadáver, no la muerte,... sino el admirable artificio de la vida". Algo después un amigo, ayudante de fisiología, le mostró, bajo el objetivo del microscopio, el movimiento circulatorio de los hematíes en el mesenterio de la rana: "Admiré por vez primera –escribirá, recordando tan sugestiva experiencia– el sorprendente espectáculo de la circulación de la sangre". Así hasta que, ya profesor, empieza a explorar por su cuenta la inmensa variedad de los paisajes histológicos del organismo animal: "se me ofrecía un campo maravilloso de observaciones, de gratísimas sorpresas... Comenzaba a deletrear con delectación el admirable libro de la organización microscópica del cuerpo humano", recordará luego con nostalgia. Delectación y sorpresa que llegarán a su cima ante la textura final del sistema nervioso, "esa obra maestra de la vida", y definitivamente cristalizarán en el "culto al cerebro" de nuestro máximo sabio.

Sucesivamente han asombrado a Cajal la naturaleza, el artificio técnico, el poder de la ciencia, el cuerpo humano. Acabamos de verlo. Pero la descripción de esa ascendente escala de los asombros cajalianos no quedaría completa si no se hiciese notar en ellos la existencia de una secreta dimensión ve-



2. El ajedrez fue pasatiempo predilecto de Cajal. En la foto, sacada por uno de sus hijos, aparece con su amigo Federico Olóriz jugando una partida durante las vacaciones de verano de 1898 en Miraflores de la Sierra.

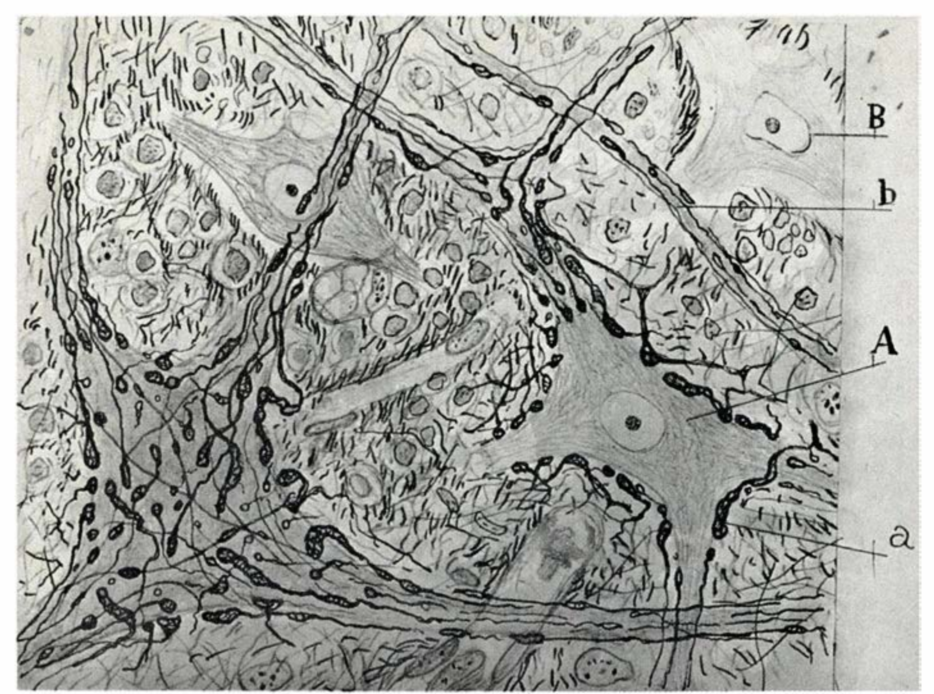
nerativa. "En el fondo de él –del mundo visible– todo es arcano, misterio y maravilla", escribe. Lo cual le llevará a hacer suya esta frase, venerativa también, de Geoffrey Saint-Hilaire: "Delante de nosotros está siempre el infinito". "*Je ne vois qu'infini par toutes les fenêtres*", escribió Baudelaire. Desde su ventana preferida, el ocular del miscroscopio, eso es también lo que con los ojos de su espíritu veía Cajal.

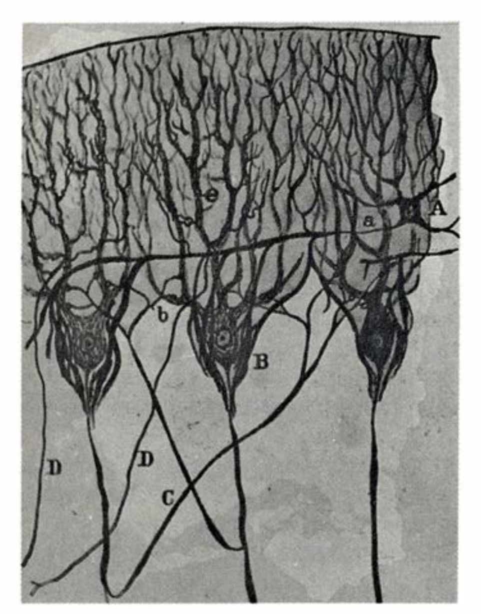
El sabio no llegaría a hacer ciencia si ante el espectáculo del mundo sólo asombro y veneración surgiesen en su alma. La pura veneración puede engendrar pensamientos y sentimientos de índole religiosa, no ideas y saberes de carácter científico. "¡Desgraciado el que en presencia de un libro –o de un espectáculo de la naturaleza– queda mudo y absorto!", dice Cajal, y agrega: "La veneración excesiva, como todos los estados pasionales, excluye el sentido crítico". Para que el asombro sea fuente de saber científico es preciso que rápidamente se resuelva en extrañeza y que ésta se articule como interrogación.

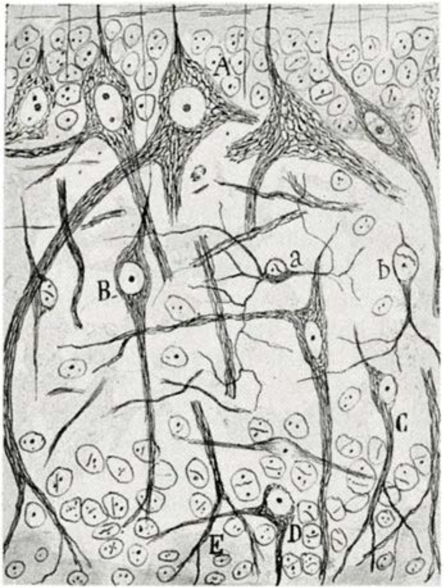
Muy precozmente lo advirtió en sí mismo Cajal. Fue en Valpalmas, con ocasión del eclipse de sol del año 1860. He aquí las palabras con que el sabio recuerda esa infantil experiencia intelectual: "Mi espíritu flotaba en un mar de confusiones, y las interrogaciones angustiosas se sucedían sin hallar respuesta satisfactoria... El saber humano,... ¿gozará del singular privilegio de comprender y vaticinar lo lejano, aquello que menos puede interesarnos desde el punto de vista de su utilidad material? Claro que estas interrogaciones –añade cautamente el autobiógrafo– no fueron pensadas en esta forma; pero ellas traducen bien, creo yo, mis sentimientos de entonces".

Los textos de Cajal, cuando recuerda y describe algunos de los pasos importantes en el curso de su larga vida científica –su trabajo sobre el mecanismo de la inflamación purulenta, primero de los que publicó; las observaciones previas a su ruptura con el reticularismo de Gerlach y Golgi, ya en la cima de su vida; su ulterior decisión de explorar por sí mismo la textura del córtex cerebral–, muestran muy claramente cómo el tránsito del asombro a la interrogación fue en él experiencia intelectual constante. "Sólo las cabezas sencillas o las ayunas de curiosidad filosófica o científica –dirá más tarde, comentando la compleja y eminente personalidad del cirujano Alejandro San Martín–, gozan del reposo y la fe. Al modo del aire en las cordilleras, en los espíritus elevados el pensamiento está en perpetua inquietud." Luego veremos lo que esa perpetua inquietud del pensamiento fue para Cajal.

A la extrañeza y la interrogación en que se resuelve el inicial asombro del sabio da primera respuesta la idea ex-







3. Tres de los dibujos histológicos de Cajal. **Arriba:** *Plexos nerviosos pericelulares de la substancia gris reticular en el bulbo de un conejo adulto, según el método del nitrato de plata reducido.* **Abajo, izquierda:** *Esquema que muestra algunas de las arborizaciones terminales libres reveladas en el cerebelo por la técnica argéntica; a la* **derecha**, *neurofibrillas de las células mitrales y en penacho del bulbo olfativo de un conejo adulto, según el método del nitrato de plata reducido.*

plicativa que a título de hipótesis pronto surge en su mente. Tal respuesta, "primer balbuceo de la razón en las tinieblas de lo desconocido", dice de ella Cajal, ha recibido varios nombres. Claudio Bernard la llamó "idea *a priori*". Siguiendo al biólogo Weissmann, nuestro sabio prefiere llamarla "hipótesis de trabajo", y ve en ella "una interrogación interpretativa de la naturaleza". Forma parte de la investigación misma, como que constituye su fase inicial", y tiene dos dimensiones principales, una intelectiva, en cuanto que, siquiera sea conjetural y provisionalmente, permite entender la realidad frente a la cual ha surgido, y otra operativa, en cuanto que incita a confirmar o a rechazar, por la vía de la observación y el experimento, la explicación que ella ofrece. "La hipótesis de trabajo y el dato objetivo –dice Cajal– están ligados por estrecha relación etiológica. Aparte su valor conceptual o explicativo, aquélla entraña un valor instrumental".

¿Cómo suscitar la aparición de hipótesis de trabajo en la mente del aspirante a investigador? Leyendo atentamente la obra de Cajal, hasta seis reglas metódicas es posible discernir: lectura atenta de lo que sobre la materia se sabe y voluntad de revivir en uno mismo el estado de espíritu por que atravesaron quienes edificaron ese saber; constante disposición de la mente para descubrir errores y limitaciones en los hallazgos ajenos; contemplación instante y amorosa de las cosas observadas: "no basta examinar, hay que contemplar con emoción y simpatía", dice textualmente Cajal; cuidado de dar al propio espíritu cierta formación filosófica; cultivo de la capacidad para cambiar de opinión, tan pronto como la realidad lo exija; extensión metódica del campo de lo observado hacia otros análogos a él o de él homólogos. Quien así proceda no tardará en descubrir que "no hay cuestiones agotadas, sino hombres agotados en las cuestiones", porque la naturaleza "nos reserva a todos, grandes o chicos, extensiones inconmensurables de tierras ignotas".

Atenido a su propia experiencia, afirmó Claudio Bernard que "la idea *a priori* surge en la mente del sabio con la rapidez del relámpago, como una revelación". Revelación: tal es justamente la palabra que emplea Cajal para nombrar el modo como las más fecundas y certeras hipótesis de trabajo vinieron a su espíritu. Un solo ejemplo, relativo a la primera intuición del máximo logro intelectual de su vida, la concepción neuronal del tejido nervioso. Aconteció en Barcelona, el año 1888: "Declaro –nos dice el sabio– que la *nueva verdad*, laboriosamente buscada, y tan esquiva durante dos años de vanos tanteos, surgió de repente en mi espíritu como una revelación". Una pregunta se impone: ¿de dónde y cómo surgen en la mente del sabio las fulgurantes "revelaciones" con que se inicia el descubrimiento de una nueva verdad? ¿Qué relación existe entre ellas y la "vivencia del ajá" que en sus chimpancés inventores hace tantos años describió el psicólogo Köhler? Graves cuestiones, que ahora debo limitarme a formular.

II. *Sentido y consistencia del saber científico.* Asombro, interrogación, hipótesis de trabajo como revelación súbita; tales son las etapas iniciales de la investigación científica. Ya en posesión de una hipótesis de trabajo verdaderamente plausible, y mediante las técnicas operatorias que posee o inventando y aplicando otras nuevas, el investigador interroga a la realidad acerca del acierto o el error de su conjetura; y en el caso de que la observación y el experimento la confirmen, eleva su hi-

pótesis a tesis y enriquece con una verdad nueva el tesoro del saber humano.

Cien veces procedió así Cajal, y el resultado fue la abundantísima cosecha de hechos e ideas con que incrementó el conocimiento morfológico y funcional del sistema nervioso. No sería pertinente aquí una enumeración de ellos; tanto menos, cuanto que el propio Cajal expuso los principales en *Recuerdos de mi vida*, y más minuciosa y sistemáticamente los registró su fiel discípulo Tello, un año después de la muerte del maestro. Yo sólo quiero descubrir y mostrar el sentido y la consistencia que para nuestro sabio tuvo su propio saber y, por extensión, el saber científico en general.

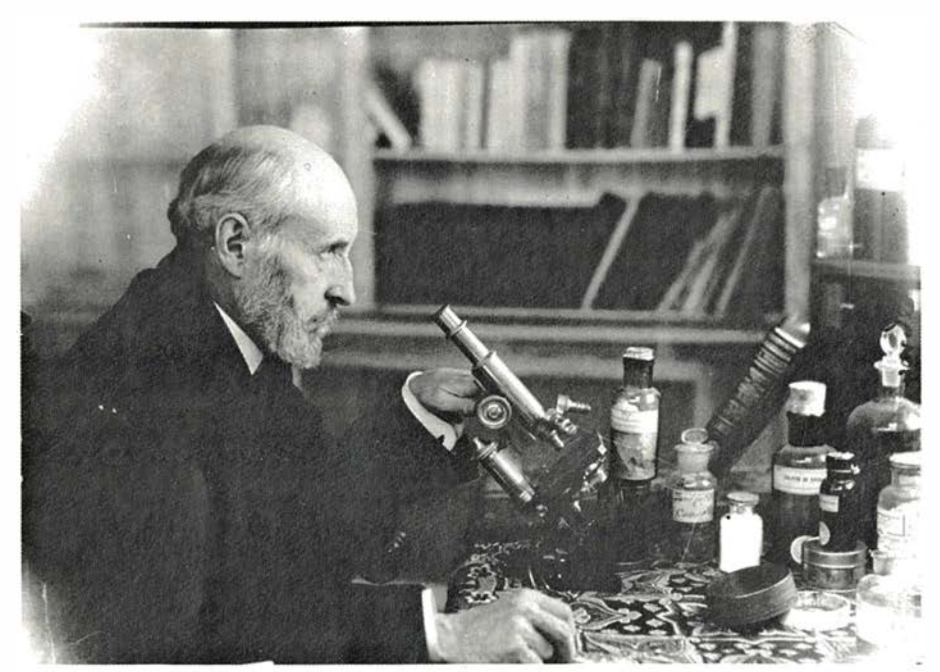
Para el hombre Santiago Ramón y Cajal, ¿qué significaron las verdades por él descubiertas? Adelantaré la respuesta: con sus hipótesis de trabajo y los descubrimientos a que ellas condujeron, Cajal –como él, *mutatis mutandis*, todo investigador– buscaba para su ser personal un estado más perfecto, más acabado, más alto que aquel en que existía antes de iniciar su investigación; y con él, la relativa, pero muy real felicidad de *ser más*. No tomada esta concisa expresión en su sentido meramente psicológico, el "ser más" de quien es más sabio o más hábil que otros, ni en su sentido meramente sociológico, ese a que se alude cuando se afirma que en una sociedad "es más" el opulento que el miserable, sino en un sentido globalmente antropológico y ontológico –ser más alta y acabadamente hombre–, tal es la meta de todo aquel que en cualquier actividad aspira a la obtención de un resultado a la vez original y valioso. Nuestro problema consiste en saber cómo entendió Cajal esa mayor perfección de su ser de hombre.

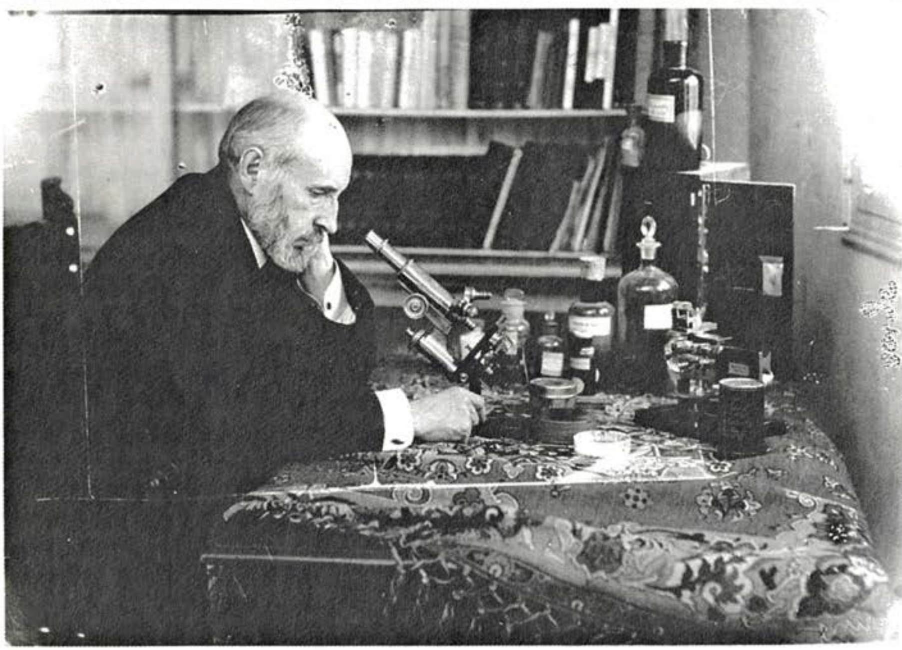
En tres asertos, complementariamente conexos entre sí, puede ser ordenada la respuesta.

1.º El sentido más radical del saber científico consiste en levantar al que lo consigue a una más alta dignidad humana. Una satisfacción esencialmente egoísta acompaña, por tanto, a la conquista y la posesión de ese saber. Entre bromas y veras, dice Cajal en uno de sus *Cuentos de vacaciones*: "El sabio posee mentalidad eminentemente aristocrática. Los que le conocen únicamente por sus obras creen –inocentes– que trabaja para la Humanidad. ¡No tal: labora para su orgullo! El investigador ama el progreso... hecho por él".

Este medular egoísmo del investigador, existente hasta en quienes del modo más notorio se sacrifican por los demás, porque también el héroe y el santo persiguen su propia perfección, es enteramente equiparable al sentimiento del descubridor de tierras nuevas: "Estoy persuadido –declara en otro lugar– de que la verdadera originalidad se halla en la ciencia: el afortunado descubridor de un hecho importante es el único que puede lisonjearse de haber hollado un terreno completamente virgen". Pero en el caso del sabio, ¿no tendrá más hondo sentido tal distinción? La respuesta de Cajal es tan tajante como solemne: "La nobleza del sabio consiste en ser ministro del progreso, sacerdote de la verdad y confidente del Creador... Al sabio solamente le ha sido dado desentrañar la maravillosa obra de la Creación para rendir a lo Absoluto el culto más grato y acepto: el de estudiar sus portentosas obras, para en ellas conocerle, admirarle y reverenciarle".

En sus dos últimos siglos, el mundo occidental ha alumbrado dos tipos de sabio. En el siglo XIX, el sabio-sacerdote; ese que está convencido de ofrecer a la humanidad lo que para ella debe ser *último*, y de procurarle, por tanto, el verdadero fundamento de su





4. Autorretratos ante el microscopio, hacia 1920.

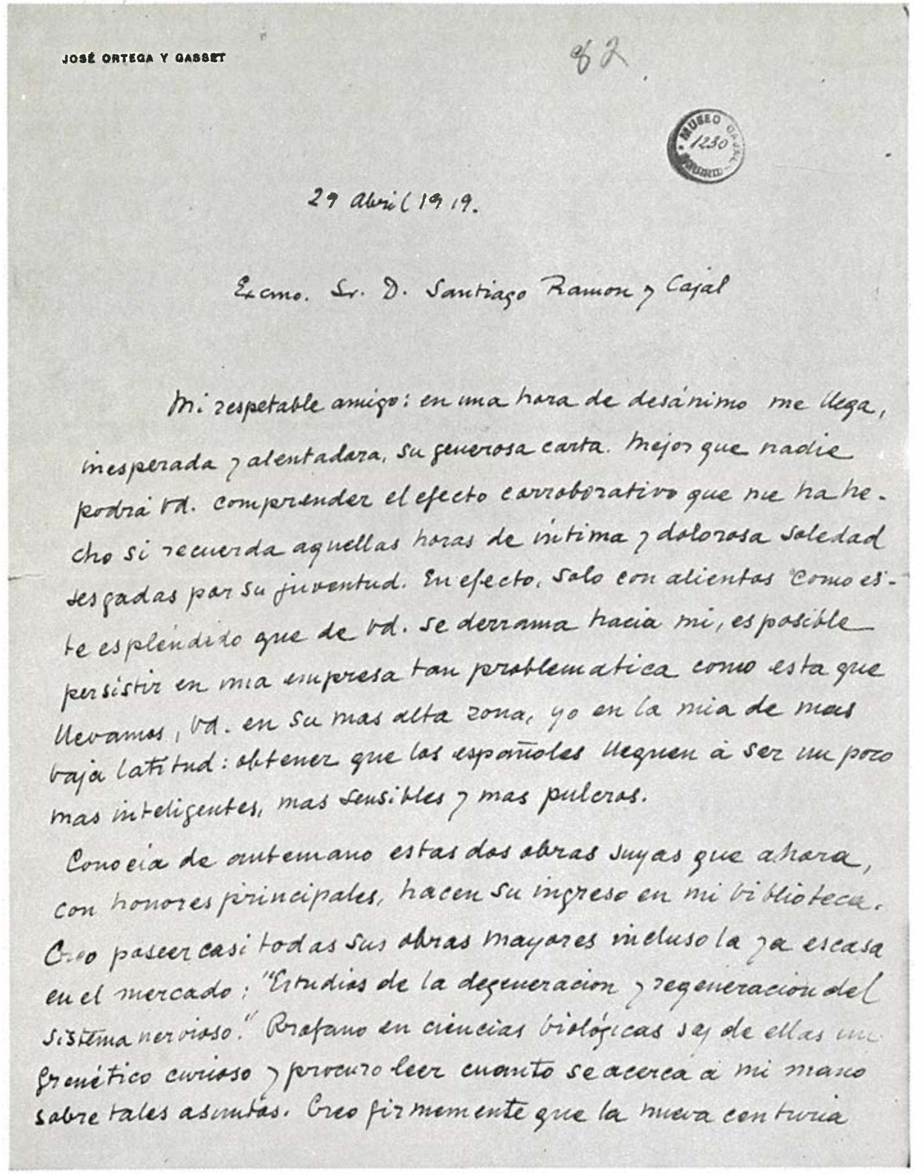
vida. En el siglo xx, el sabio-deportista; ese que, aun sabiendo que como hombre de ciencia se está moviendo en el campo de *lo penúltimo*, porque sólo penúltimo puede ser para el hombre entero el saber científico, a éste consagra su vida, incluso con sacrificio y con riesgo, si esto fuese necesario.

2.º Por ser como es la existencia del hombre, el plus de dignidad humana logrado por el sabio revierte necesariamente en favor de los demás hombres, aunque éstos no lleguen a percibirlo. "Si prescindimos del íntimo resorte egoísta que mueve a la inteligencia investigadora y consideramos los efectos sociales de cada descubrimiento, la pretensión altruista del sabio se confirma: sus investigaciones benefician positivamente a la Humanidad." Así sucede en la actividad amorosa: "En ciencia, como en amor, el protagonista es engañado por la naturaleza. En virtud de una ilusión irremediable, el sabio y el amante creen, tocante a sus respectivas actividades, trabajar *pro domo sua*, cuando en realidad no hacen sino obrar en provecho y gloria de la especie".

3.º Situada entre la persona individual y la humanidad, la patria es la estructura de la coexistencia humana en que de modo más directo y eminente se realiza la comunicación social de la excelencia y el progreso logrados por el sabio. "Quien piensa y siente fuerte –escribía nuestro histólogo en 1898– envejece y gasta sus energías cerebrales... Pero por una compensación muy sabia, lo que el individuo gasta en labor mental propia lo benefician la especie, la raza y la nación." Especie, raza y nación. En el pensamiento y en el lenguaje de Cajal, esos son, con la familia, los órdenes principales de la coexistencia humana.

Con la palabra *especie*, Cajal alude, como es obvio, a la general humanidad, al conjunto de los hombres todos. ¿Y qué es lo que la recoleta obra del sabio puede otorgar a la especie humana? Para nuestro meditabundo histólogo, cuatro bienes distintos: dignidad, holgura vital, poderío y esperanza terrena. "Lucha el sabio en beneficio de la Humanidad entera –dijo en ocasión solemne–, ya para aumentar y dignificar la vida, ora para acallar el dolor, ora para retardar y dulcificar la muerte."

Los términos *raza* y *nación* suelen ser usados por Cajal –que vivió y se formó cuando el cansancio histórico de los españoles les hacía oponer un entre amargo y esperanzado "patriotismo de la raza" al habitual "patriotismo de la historia" o "de la tradición"– con significación poco precisa y muy semejante. No puedo estudiar ahora cómo una veta sentimental, otra crítica y otra operativa se entrelazan de vario modo, a lo largo de la vida de Cajal, en la estructura de su siempre ardoroso patriotismo. Debo conformarme recordando dos textos especialmente expresivos de su constante manera de entender la vinculación entre su obra científica y su condición de español. En el discurso de respuesta al homenaje de su universidad cuando le fue concedido el Premio de Moscú, declaró con vehemencia que era permanente ideal de su alma "aumentar el caudal de ideas españolas circulantes por el mundo, granjeando respeto y simpatía para nuestra ciencia, colaborando, en fin, en la grandiosa empresa de descubrir la naturaleza, que es tanto como descubrirnos a nosotros mismos". *A patria chica, alma grande*, fue el título de ese discurso. Cinco años más tarde, toda la España culta vibró con la celebración del tercer centenario del *Quijote*. Se diría que, tras la enorme depresión que en tantos y tantos españoles produjo el desastre de 1898, había surgido en España una sorda necesidad colectiva de afirmar la valía insigne de nuestra máxima hazaña literaria y la alta incitación ética que en ella late. Cajal, por lo menos, la sintió con fuerza en su alma, y en una conferencia memorable nos dijo cuál debería ser la lección moral del *Quijote* para la España del siglo xx: el *quijotismo del trabajo científico y técnico*, la abnegada empresa colectiva de regenerar mediante la inteligencia y el esfuerzo la



JOSÉ ORTEGA Y GASSET

62

MUSEO CAJAL 1250

29 Abril 1919.

Excmo. Sr. D. Santiago Ramon y Cajal

Mi respetable amigo: en una hora de desánimo me llega, inesperada y alentadora, su generosa carta. Mejor que nadie podrá Vd. comprender el efecto corroborativo que me ha hecho si recuerda aquellas horas de íntima y dolorosa soledad desgadas por su juventud. En efecto, solo con alientos como este espléndido que de Vd. se derrama hacia mi, es posible persistir en una empresa tan problematica como esta que llevamos, Vd. en su mas alta zona, yo en la mia de mas baja latitud: obtener que los españoles lleguen á ser un poco mas inteligentes, mas sensibles y mas pulcros.

Conocia de antemano estas dos obras suyas que ahora, con honores principales, hacen su ingreso en mi biblioteca. Creo poseer casi todas sus obras mayores incluso la ya escasa en el mercado: "Estudios de la degeneracion y regeneracion del sistema nervioso." Profano en ciencias biológicas soy de ellas un frenético curioso y procuro leer cuanto se acerca á mi mano sobre tales asuntos. Creo firmemente que la nueva centuria

5. Carta de José Ortega y Gasset. En ella dice el filósofo: "En efecto, sólo con alientos como este espléndido que de Vd. se derrama hacia mí, es posible persistir en una empresa tan problemática como esta que llevamos, Vd. en su más alta zona, yo en la mía de más baja latitud: obtener que los españoles lleguen a ser un poco más inteligentes, más sensibles y más pulcros.

decaída vida nacional. Ochenta años después de proclamada, sigue siendo actual esa noble consigna.

Los precedentes asertos expresan el pensamiento de Cajal acerca del sentido del saber científico. Ahora bien: ese vario sentido, ¿qué consistencia real posee? Respecto de la realidad misma –la del hombre, la de todo lo existente–, ¿qué es realmente el saber que el científico conquista?

A mi modo de ver, dos tesis complementarias se integran en la respuesta de Cajal:

1.ª Cuando es un hecho científico correctamente observado o una ley correctamente establecida, ese saber expresa *una verdad inobjetable, y por tanto definitivamente válida*. La relación por contigüidad entre las neuronas, y no por continuidad, como afirmaba el reticularismo de Gerlach y Golgi y todos admitían, fue para Cajal un hecho científico definitivo, un incontestable conocimiento de lo que para la mente humana es la realidad de una parcela de la naturaleza; y como ese hecho morfológico, la ley de la polarización dinámica del impulso nervioso, su natural complemento fisiológico.

2.ª En cuanto saber humano, y puesto que la mente del hombre es constitutivamente finita e histórica, el saber científico no es y no puede ser otra cosa que *una pretensión de conocimiento de lo que las cosas realmente son*; por lo cual se halla intrínsecamente afectado de insuficiencia, es por esencia perfectible y, aunque su apariencia inmediata sea la afirmación, es en definitiva un conocimiento interrogativo, una tácita pregunta dirigida a la mente de quien lo ha formulado y a la de los sabios y pensadores posteriores a él.

En su discurso rectoral de 1933 –*Die Selbstbehauptung der deutschen Universität*, "La autoafirmación de la universidad alemana"– afirmó el filósofo Martin Heidegger que la pregunta es la forma suprema del saber humano. Llegado éste a su más alto nivel, "el preguntar ya no es un previo y superable escalón hacia la verdad, sino que se convierte en la forma cimera del saber".

Sepamos entender correctamente este profundo pensamiento. Consideremos para ello un determinado saber científico; por ejemplo, la ley de la gravitación universal, en su primitiva forma newtoniana o en su ulterior versión einsteiniana. Pues bien: ¿qué es en rigor *saber* la ley de la gravitación universal? ¿Afirmar algo de manera perfecta y definitiva? ¿Haber llegado a un conocimiento que ya no puede suscitar en el hombre pregunta alguna? De ningún modo. Para una mente de veras exigente y rigurosa, y sean cualesquiera las novedades que traiga la física futura, la formulación de ese saber lleva en su seno dos graves preguntas: ¿cómo tiene que estar constituida la realidad del cosmos para que uno de sus modos de manifestación sea la relación cuantitativa entre fuerzas, masas y distancias que llamamos "ley de la gravitación universal"? ¿Cómo han de estar constituidas la existencia del hombre y mi propia existencia para que la realidad del cosmos se me presente según esa ley? Preguntas ambas a las que la mente humana –tal es el drama inexorable a que con sus mudanzas va dando expresión la historia del pensamiento– nunca será capaz de dar una respuesta imperfectible y definitiva.

Veamos ahora la actitud de nuestro sabio ante este arduo problema. ¿Qué fue para Cajal su propio saber y, más generalmente, el saber científico? Cuando éste se expresa como hecho bien establecido y como ley exacta, su validez absoluta y definitiva le parece incontrovertible. "El hecho histológico de primera mano, bien descrito y presentado –escribe–, constituye algo fija y absolutamente estable, contra lo cual ni el tiempo ni los hombres podrán nada... Soy adepto ferviente de la religión de los hechos". Frente a la fugacidad de tantas hipótesis, dice en otro lugar, "ahí están inmutables, y desafiando a la crítica, los hechos bien observados". Su lema es el de Carlyle: "Dadme un hecho, y yo me postro ante él".

No hay duda: la confianza del hombre de ciencia en la razón humana y en la validez de los saberes adquiridos mediante ella es total y absoluta; y lo que hasta hoy no haya sido racional y científicamente conocido, mañana lo será. ¿Durará hasta el fin de su vida esa orgullosa confianza de Cajal en el poder y en la firmeza de la ciencia?

En lo tocante al saber científico *stricto sensu*, dos apostillas suyas obligan a una respuesta negativa. La contemplación de la circulación sanguínea en el mesenterio de la rana le hizo concebir, cartesianamente, la convicción de que los fenómenos biológicos podrían ser racional y mecánicamente explicados: "En presencia de tan sublime espectáculo sentí una revelación... Parecióme que se descorría un velo en mi espíritu... Los cuerpos vivos, me dije, son máquinas hidráulicas tan perfectas, que son capaces de reparar los desarreglos causados por el torrente que las mueve, y de producir, en virtud de la generación, otras máquinas hidráulicas semejantes". Años más tarde, Cajal añadirá a esas palabras la apostilla siguiente: "Hoy no suscribiría yo sin alguna restricción este concepto de la vida. En ella... se dan fenómenos que presuponen causas absolutamente incomprensibles". Algo análogo dirá recordando sus primeras ideas acerca de la cognoscibilidad científica de la ontogenia y la filogenia del ojo y el oído: "Cuanto más estudio la organización del ojo de vertebrados e invertebrados, menos comprendo las causas de su maravillosa y exquisitamente adaptada organización". La edad le ha hecho ver más profunda y menos jactanciosamente la realidad. "Cuanto más solitario y más metido en mí mismo, más amigo de los mitos voy siendo", escribió Aristóteles cuando ya le vencía la edad. Quería decir: "Cuanto más profundo y rico es mi saber, más necesito un relato imaginativo acerca de la entera realidad de las cosas que como filósofo conozco". A su manera, ¿no fue ésta la actitud de Cajal?

Más obvia se muestra la respuesta negativa cuando no es del saber científico *stricto sensu* de lo que se trata, sino de la primaria y global relación de la mente humana con la realidad. ¿No hemos oído decir a Cajal que el pensamiento está en perpetua inquietud cuando es verdaderamente fuerte la pasión de conocer? Y con Geoffroy Saint-Hilaire, ¿no ve siempre el infinito –esto es: algo a cuya comprensión no puede llegar nuestra mente– en todo lo que sus ojos contemplan? Filosófico o científico, el saber del hombre será siempre una pretensión nunca entera y definitivamente lograda. Cierto: la forma suprema de nuestro saber es y tiene que ser la pregunta.

Alguien dirá que la creencia en Dios, primer principio y último fundamento de la realidad, de toda realidad, puede y debe ser la respuesta definitiva a la última de las sucesivas preguntas que ante el mundo pueda el hombre formular. Ajeno a toda confesión religiosa, Cajal declaró abierta y solemnemente que a lo largo de todas sus vicisitudes intelectuales nunca dejó de atenerse a dos soberanos principios: "la existencia del alma inmortal y la de un ser supremo, rector del mundo y de la vida". Pero, por muy firme que sea, ¿puede esta creencia eximirnos de una interrogación acerca de ella misma y de lo que ella nos dice? Si no fuera así, no habría una historia de las religiones y no existiría, en el caso de la cristiana, una evolución del dogma y del pensamiento teológico. Concluiremos, pues, que todo saber científico es un aserto más o menos verdadero y más o menos profundo acerca de una parte de la realidad, limitado por dos interrogaciones, una inicial, aquella de que procede, y otra final, aquella a que conduce, determinada ésta por la siempre insuficiente relación entre él y la total realidad de la parcela del mundo a que directamente se refiere. (Pedro Laín Entralgo.)

Sistemas operativos

El sistema operativo de un computador abarca múltiples niveles de complejidad, desde los comandos introducidos a través de un teclado hasta los pormenores de la conmutación. El sistema se organiza en una jerarquía de abstracciones

Peter J. Denning y Robert L. Brown

Escriba la orden *date* (fecha) en un terminal conectado a un computador y pulse la tecla *return* (presentación). Casi inmediatamente aparecerá en la pantalla el mensaje 15 de septiembre de 1984. Pedirle la fecha puede parecer una de las preguntas más sencillas que se pueden hacer a un computador. Sin embargo, tan sencillo ruego pone en marcha un conjunto complejo de acontecimientos que implica a muchos de los recursos, de programas y circuitos, que posee el sistema. Se denomina sistema operativo al conjunto de programas que tienen la responsabilidad, entre otras cosas, de gestionar los recursos y coordinar los sucesos en un computador. El sistema operativo proporciona las ayudas y servicios necesarios para la mayoría de los programas.

Pensemos en qué debe ocurrir para contestar a una petición de la fecha. Cada vez que se escribe un carácter del comando, el teclado transmite un código al computador; éste lo recibe a través de una placa de circuitos que encierra todo lo necesario para comunicarse con el terminal. La placa almacena cada carácter en un área reservada de memoria, denominada memoria intermedia ("buffer"), y emite una señal que "interrumpe" la unidad central de proceso del computador; activa así un programa denominado controlador del terminal. El controlador del terminal devuelve una copia del carácter al terminal para que éste lo represente en la pantalla.

Cuando se recibe el código de la tecla *return*, que significa que se ha terminado de escribir el comando, el controlador del terminal activa otro programa, denominado oyente ("listener" en inglés, ya que "está a la escucha" de las peticiones de los usuarios). El oyente lee los caracteres *d a t e* almacenados en la memoria intermedia del teclado, busca en una memoria de discos magnéticos un programa denominado *date*, carga el programa en la memoria principal e inicia su ejecución. El programa *date* consulta, a su vez, un reloj construido en los circuitos, el cual mantiene una cuenta de los milisegundos transcurridos desde una fecha inicial prefijada. A partir de dicha cuenta, el programa calcula el mes, el día y el año, y expresa la información en forma de una secuencia de caracteres: 15 de septiembre de 1984. Estos caracteres se pasan al programa controlador del terminal que transmite el código binario de carácter al terminal, y éste lo visualiza en la pantalla.

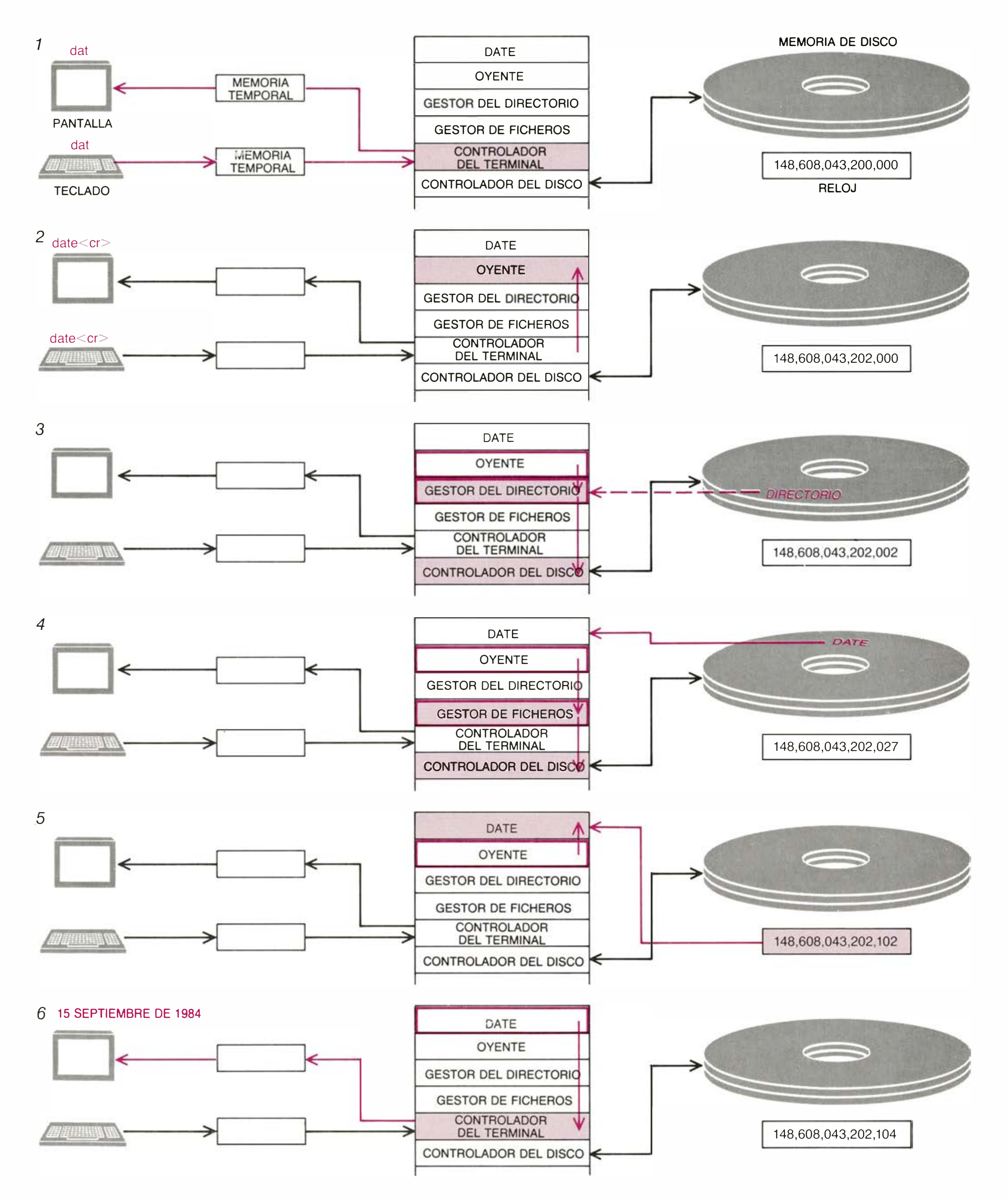
Cada suceso de éstos podría describirse con mayor detalle. Por ejemplo, antes de que el programa oyente pueda cargar el programa *date*, debe mirar primero en un directorio de comandos para descubrir en qué posición del disco se encuentra el programa; para ello tiene que leerse el propio directorio en el disco. Por estar el disco organizado en pistas concéntricas, y cada pista en sectores, deben ejecutarse instrucciones que coloquen el cabezal de lectura sobre la pista apropiada y que lean los datos binarios cuando el sector seleccionado pase por debajo del cabezal. La secuencia de bits leídos se almacena de manera transitoria en una memoria intermedia. Antes de cargar el programa, se le debe asignar espacio en la memoria principal; cuando acabe su ejecución, se recuperará ese espacio de memoria. La secuencia de sucesos es mucho más complicada en un computador que trabaje simultáneamente con varios programas. En cuyo caso, un programa puede suspenderse provisionalmente mientras el procesador central atiende a otro; después, el primer programa debe reanudar su ejecución en el punto exacto donde estaba, como si no hubiera quedado interrumpido.

Del ejemplo se desprende que el sistema operativo abarca toda la complejidad que poseen los computadores. Algunas partes del sistema operativo interactúan directamente con los circuitos del computador, donde los sucesos (tales como la conmutación de puertas lógicas individuales) pueden precipitarse a una velocidad de miles de millones de veces por segundo. En el otro extremo del espectro, algunas partes del sistema operativo se comunican con el usuario, que envía comandos a una velocidad mucho menor, uno cada varios segundos quizá. El mero hecho de apretar una sola tecla en el terminal puede originar 10 llamadas a los programas del sistema operativo, la ejecución de 1000 instrucciones de máquina y 10 millones de cambios de estado de las puertas lógicas.

La estrategia adoptada para gestionar esta complejidad ha demostrado ser de crucial importancia en la inmensa mayoría de ámbitos de la informática. La idea central consiste en crear una jerarquía de niveles de abstracción de forma que, en cualquiera de ellos, se pueden ignorar los detalles de lo que ocurre en los niveles inferiores. Así, cuando el programa oyente carga un programa desde el disco de memoria, el oyente no necesita especificar la ubicación del cabezal de lectura; dichas operaciones mecánicas dependen de un programa situado en un nivel inferior de la jerarquía. En el máximo nivel superior se encuentra el usuario del sistema, idealmente aislado de cualquier cosa, excepto de lo que intenta realizar.

Los sistemas operativos se crearon para los computadores electrónicos que comenzaron a aparecer a finales de los años cuarenta. Estaban formados por simples rutinas para entrada y salida, tales como programas para almacenar códigos binarios, leídos desde una cinta de papel perforado, en posiciones sucesivas de memoria. El sistema operativo completo constaba de algunos centenares de instrucciones de máquina.

A mitad de los años cincuenta, mu-

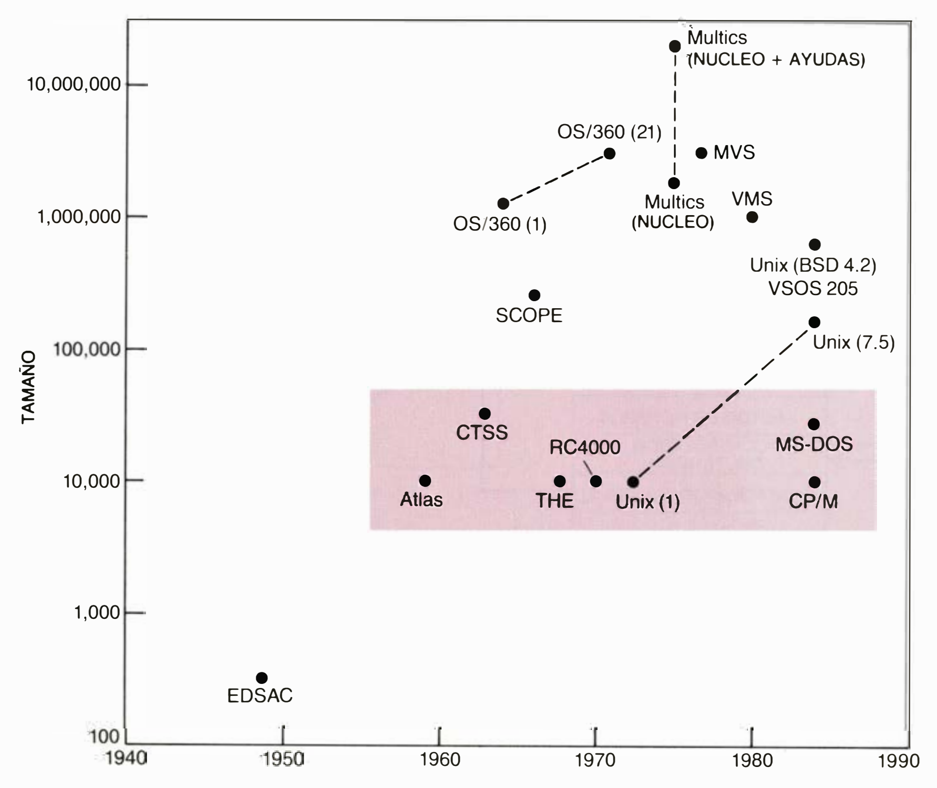


1. LA EJECUCION DE UN COMANDO pone en acción sucesos de distintos niveles de la jerarquía de programas que constituyen el sistema operativo. El comando considerado es la petición de la fecha. Cada carácter que se escribe en el teclado (*1*) se recibe en el programa que controla el terminal, que lo remitirá a la pantalla. Cuando se introduce el carácter de retorno, el controlador del terminal pasa la cadena de caracteres *d a t e* (*fecha*) al programa oyente (*2*), que lo interpreta como el nombre de un comando. El oyente solicita del gestor del directorio que busque en él el comando *date*. A su vez, el gestor del directorio solicita al gestor del disco que copie el directorio en una memoria intermedia (buffer) dentro de la zona de trabajo del gestor del directorio (*3*). Cuando se encuentra el comando, el oyente instruye al gestor de ficheros para que cargue en memoria el código binario del programa *date*; para realizar esto, el gestor de ficheros usa de nuevo el controlador del disco (*4*). El oyente activa después el programa *date*, y éste lee un "reloj" (*5*), dispositivo formado por circuitos que mantiene una cuenta de los milisegundos que han transcurrido desde alguna fecha inicial fija, en este caso la medianoche del 1 de enero de 1980. A partir de esta cuenta, el programa calcula la fecha actual y la visualiza a través del gestor del terminal como 15 de septiembre de 1984 (*6*). El oyente y los diversos gestores y controladores forman parte del "núcleo" del sistema operativo; *date* es un programa de ayuda al usuario.

chos computadores trabajaban "en lotes". Un sistema operativo recogía los programas enviados por múltiples usuarios y los ejecutaba en secuencia, eliminando así los retrasos que se producían al cargar los programas uno a uno. Los sistemas operativos de este tipo fueron llamados supervisores o monitores. Además de su función primaria de carga de programas, gestionaban los dispositivos de almacenamiento secundario (discos, tambores y cintas magnéticas), asignaban memoria principal y realizaban las entradas y salidas. En la mayoría de los casos, poseían también una "biblioteca" de programas, con las rutinas más frecuentemente utilizadas. Por ejemplo, muchas aplicaciones de computador necesitan clasificar la información; si la biblioteca posee una rutina versátil para clasificar, el sistema operativo puede cargarla junto con cada programa que la necesite.

Los primeros sistemas de tiempo compartido fueron diseñados alrededor de 1960. En este método de trabajo, la atención del procesador central cambia rápidamente entre los diferentes programas de los usuarios, dándoles la sensación de que se ejecutan simultáneamente. Al construir tales sistemas, se deben resolver los problemas que entraña compartir el procesador, la memoria y los distintos recursos logicales. La resolución de dichos problemas dio lugar a un número importante de notables avances conceptuales, entre ellos la sincronización de procesos paralelos, la memoria virtual, la entrada y salida independientes del dispositivo y los lenguajes de comandos interactivos, de todos los cuales hablaremos.

Cuanto más elaborado es un sistema operativo, mayor tamaño adquiere. El Compatible Time Sharing System, que se puso en funcionamiento en el Instituto de Tecnología de Massachusetts, en 1963, constaba de unas 32.000 palabras de 36 bits. El sistema os/360, introducido un año más tarde por IBM, posee más de un millón de instrucciones de máquina. En 1975, el sistema Multics, desarrollado por el MIT y los laboratorios de la Bell, poseía ya más de 20 millones de instrucciones.



2. EVOLUCION DE LOS SISTEMAS OPERATIVOS. Sugiere una tendencia al crecimiento exponencial, aunque recientemente se han introducido sistemas más pequeños para los microcomputadores. El tamaño se da en unidades que corresponden a las "palabras" de memoria de la máquina, o a instrucciones del lenguaje máquina (lenguaje ensamblador). El EDSAC fue desarrollado en la Universidad de Cambridge, Atlas en la Universidad de Manchester, CTSS en el Instituto de Tecnología de Massachusetts, THE en la Universidad Politécnica de Eindhoven y RC4000 en la Universidad de Dinamarca. SCOPE es un producto de Control Data Corporation, así como VSOS 205; OS/360, MVS y VMS, por su parte, los fabrica la compañía IBM. MULTICS fue desarrollado conjuntamente por el MIT y los laboratorios Bell; Unix, por los laboratorios Bell. CP/M y MS-DOS son sistemas operativos para microcomputadores introducidos, respectivamente, por Digital Research Inc. y Microsoft Corporation. Los diferentes sistemas que aparecen encerrados dentro de la banda en color son núcleos para un sola máquina, probablemente de tamaño mínimo.

En esa época, los minicomputadores habían entrado con éxito en el mercado y los microcomputadores (ordenadores personales incluidos) acababan de aparecer, compensando así la tendencia al crecimiento de los sistemas operativos. Estas máquinas eran más lentas y poseían una menor capacidad de memoria que los grandes computadores de la época, pero acercaban el ordenador a un número muy superior de usuarios potenciales. Para adaptar los sistemas operativos al menor espacio de los minis y micros, se dividieron las funciones del sistema operativo. Los servicios requeridos por la mayoría de los programas, tales como las rutinas de entrada y salida, se colocaron en un "núcleo" que permanece en la memoria principal del computador siempre que éste se halle trabajando. Otros programas, denominados ayudas del sistema, están ubicados en el disco y se traen a la memoria principal solamente cuando se necesitan. A juzgar por los sistemas operativos introducidos en los últimos años, parece que el núcleo mínimo necesario para gestionar los recursos de un computador contiene unas pocas decenas de millares de instrucciones. Las ayudas adicionales y las bibliotecas de programas crecen continuamente, de manera casi exponencial, forzando la capacidad de las disponibilidades de almacenamiento secundario.

La evolución de los sistemas operativos no ha terminado. Una serie de nuevos usuarios, de los cuales muchos ni siquiera se dedican por entero a los computadores, plantean la exigencia de nuevos servicios de programación. Se ha respondido con el desarrollo de despliegues gráficos interactivos. Con dichos despliegues, se puede borrar un fichero, no escribiendo la orden *delete* (borrar), sino apuntando al dibujo de una papelera. Han aparecido, asimismo, nuevas formas de organización de un computador. En vez de disponer de un único computador grande, conectado a muchos terminales, cada usuario puede tener su estación de trabajo, con un procesador propio, comunicándose con otras estaciones de trabajo a través de una red de alta velocidad. En tales redes de procesado distribuido, es responsabilidad del sistema operativo la coordinación de las acciones de los diversos computadores existentes.

La estructura jerárquica de un sistema operativo moderno separa sus funciones de acuerdo con su complejidad, su escala temporal característica y su nivel de abstracción. La figura 3 mues-

tra una organización que abarca trece niveles. No es un modelo de un sistema concreto, sino que incorpora ideas de varios de ellos; cuenta también con dispositivos para el procesado distribuido. Cada nivel gestiona un conjunto de "objetos", que pueden ser circuitos o programas, y cuya naturaleza varía considerablemente de un nivel a otro. Cada nivel define también las operaciones que pueden realizarse sobre los objetos.

Los niveles más bajos incluyen los circuitos del sistema. El nivel 1 corresponde a los circuitos electrónicos, donde los objetos son registros, celdas de memoria, puertas lógicas y componentes análogos. Las operaciones definidas sobre estos objetos son acciones tales como poner a cero un registro o leer una posición de memoria. El nivel 2 corresponde al conjunto de instrucciones del procesador, que puede operar con algunas abstracciones, tales como pilas de evaluación (secuencias de registros o celdas de memoria donde se guardan los valores numéricos a la espera de que se realice alguna operación con ellos). Las operaciones a este nivel son las instrucciones que el procesador por sí mismo puede ejecutar: *sumar* (add), *restar* (substract), *cargar* (load) y *almacenar* (store), entre otras.

El nivel 3 añade el concepto de procedimiento o subrutina, un fragmento de programa autocontenido que puede reclamarse desde el interior de otro programa mayor y que devuelve el control al punto desde donde fue llamado. El nivel 4 introduce las interrupciones, que permiten que el procesador guarde un registro de su estado actual y se ponga a ejecutar otra tarea. Entre los sucesos que generan una interrupción están ciertos tipos de errores, así el desbordamiento en un registro aritmético, y sucesos más comunes, tales como la recepción de un carácter desde un terminal.

El conjunto de los cuatro primeros niveles corresponde, básicamente, a la máquina tal como la entrega el constructor, aunque existen interacciones íntimas con algunos elementos del núcleo del sistema operativo. Por ejemplo, las interrupciones son generadas por los circuitos, pero las rutinas que se llaman cuando se interrumpe al procesador son parte del núcleo.

NIVEL	NOMBRE	OBJETOS	EJEMPLO DE OPERACIONES
13	CAPARAZON	ENTORNO DE PROGRAMACION DEL USUARIO	INSTRUCCIONES EN LENGUAJE DE CAPARAZON
12	PROCESOS DE USUARIO	PROCESOS DE USUARIO	ABANDONAR, BORRAR, SUSPENDER, REANUDAR
11	DIRECTORIOS	DIRECTORIOS	CREAR, DESTRUIR, ASIGNAR, DESASIGNAR, BUSCAR, LISTAR
10	DISPOSITIVOS	DISPOSITIVOS EXTERNOS: IMPRESORAS, PANTALLAS, TECLADOS,...	CREAR, DESTRUIR, ABRIR, CERRAR, LEER, ESCRIBIR
9	SISTEMA DE FICHEROS	FICHEROS	CREAR, DESTRUIR, ABRIR, CERRAR, LEER, ESCRIBIR
8	COMUNICACIONES	CANALES	CREAR, DESTRUIR, ABRIR, CERRAR, LEER, ESCRIBIR
7	MEMORIA VIRTUAL	SEGMENTOS DE MEMORIA	LEER, ESCRIBIR, EXTRAER
6	MEMORIA LOCAL AUXILIAR	BLOQUES DE DATOS, CANALES PERIFERICOS	LEER, ESCRIBIR, ASIGNAR, LIBERAR
5	PROCESOS PRIMITIVOS	PROCESOS PRIMITIVOS, SEMAFOROS, LISTA DE DISPONIBLES	ESPERAR, SEÑALAR, SUSPENDER, REANUDAR
4	INTERRUPCIONES	PROGRAMAS PARA MANEJO DE ERRORES	LLAMAR, ENMASCARAR, DESENMASCARAR, INTENTAR DE NUEVO
3	PROCEDIMIENTOS	SEGMENTOS DE PROCEDIMIENTO, PILA, VISUALIZACION	SEÑALAR LA PILA, LLAMAR, DEVOLVER
2	CONJUNTO DE INSTRUCCIONES	PILA DE EVALUACION, INTERPRETE DE MICROPOGRAMA, DATOS ESCALARES, DATOS VECTORIALES	CARGAR, ALMACENAR, ROMPER SECUENCIA, SUMAR, RESTAR
1	CIRCUITOS ELECTRONICOS	REGISTROS, PUERTAS, BUSES, ETC.	LIMPIAR, TRANSFERIR, ACTIVAR, COMPLETAR

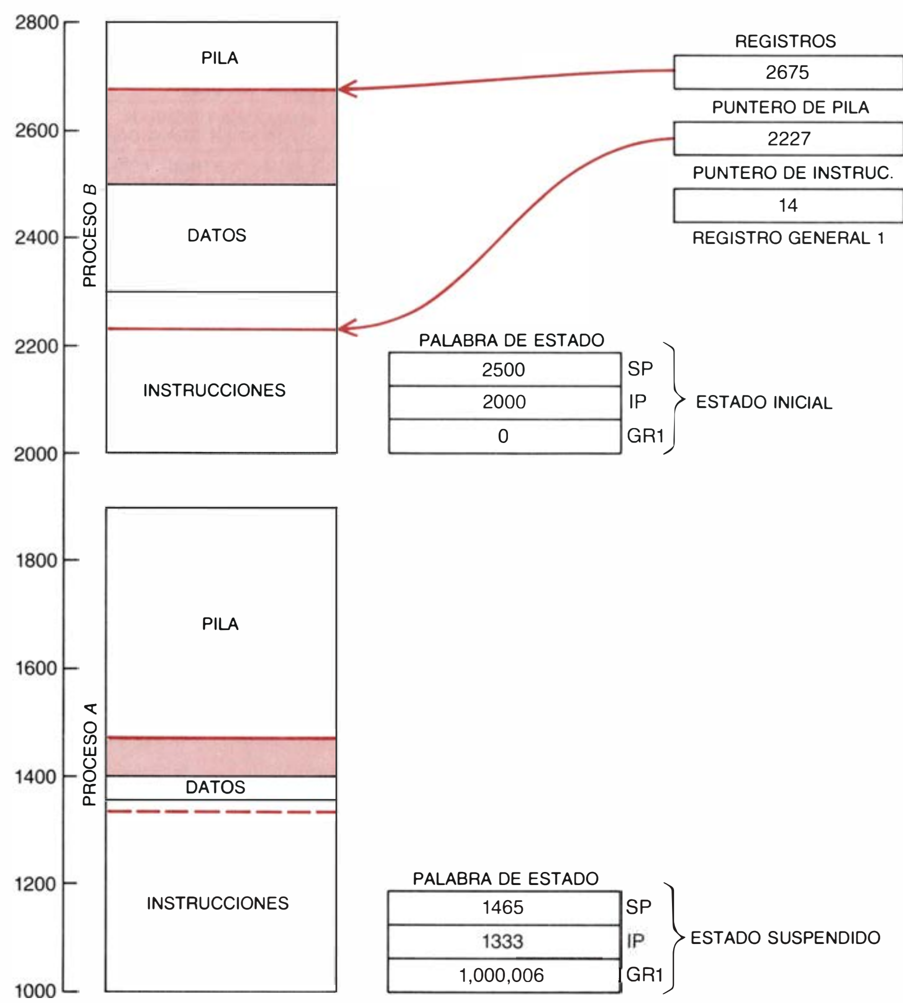
3. EN LA JERARQUIA DE ABSTRACCIONES reside el principio fundamental para organizar un sistema operativo. Cada nivel es el gestor de ciertos "objetos", circuitos o programas. Un programa de un nivel dado sólo tiene acceso a las operaciones definidas en los inferiores al suyo; además, los detalles internos de estas operaciones no son visibles. Los siete niveles más bajos están relacionados con las operaciones de la máquina; los niveles altos traban en una red los recursos de varios computadores.

Los conceptos asociados explícitamente con la coordinación de tareas múltiples aparecen, por primera vez, en el nivel 5, identificado como el nivel de "procesos básicos" o programas singulares en curso de ejecución. Puesto que un proceso primitivo puede interrumpirse en cualquier instante, es necesario un mecanismo para suspender un proceso y reanudarlo después. El mecanismo consiste en una "palabra de estado", una estructura de datos que puede almacenar el contenido de todos los registros del procesador central más una operación de "cambio de contexto". Cuando debe pararse un proceso, la operación de cambio de contexto copia los valores de los registros en la palabra de estado de ese proceso; para reanudarlo, restaura los registros con sus valores primitivos.

Si todas las actividades que se desarrollan dentro de un computador fueran independientes unas de otras, se necesitaría poco más que la noción de palabra de estado para crear un sistema operativo multiproceso. En realidad, un proceso depende frecuentemente de los resultados obtenidos por otro, por lo que los procesos deben estar sincronizados. Por ejemplo, un programa que requiera datos de un fichero del disco no puede proceder hasta que los datos hayan sido leídos y estén disponibles en la memoria principal. Es imposible que el programador conozca de antemano cuánto tardará en realizarse la operación de lectura en el disco; por lo tanto, es necesario un procedimiento que permita que un programa espere hasta que otro le avise que ha llegado su turno.

El concepto que aporta la clave de la sincronización es el semáforo; ocupa un lugar primordial en la teoría de los sistemas operativos. En su sentido inmediato, podemos imaginar el semáforo como la señal de ferrocarril, con sus luces roja y verde, indicando si un proceso tiene o no permiso para continuar. En el punto donde un proceso debe sincronizarse con alguna rutina externa, el programador inserta la instrucción *wait* (*semaphore A*) -espere (semáforo A). Cada vez que se llega a la ejecución de ese punto del programa, se inspecciona el semáforo. Si está en rojo, se suspende la ejecución del proceso; si está verde, el proceso continúa pero el semáforo se pone en rojo. Cuando el segundo proceso lanza una instrucción *signal*(*semaphore A*), el semáforo vuelve a ponerse en verde y el primer proceso prosigue la ejecución, caso de estar aguardando.

En realidad, y puesto que varios procesos deben controlarse con el mismo semáforo, el semáforo debe encerrar mayor complejidad: contendrá un contador y una cola de procesos en espera. Por cada instrucción *wait* el contador decrece, y crece por cada instrucción



4. PROCESOS PRIMITIVOS: representan un solo programa en curso de ejecución. Aquí se muestran dos de ellos cargados en un segmento de la memoria principal. El proceso *B* está en fase de ejecución. El puntero de instrucciones, uno de los registros internos del soporte físico del computador, indica la dirección de la siguiente instrucción; el puntero de pila señala la posición del último dato introducido en un área de almacenamiento temporal denominada pila. Un computador real debería tener muchos más registros de propósito general, aunque en este esquema diagramático sólo se muestra uno. El proceso *A* ha sido suspendido, pero los contenidos de todos los registros en el momento de la suspensión han sido grabados en un área reservada de memoria denominada palabra de estado. El sistema operativo puede conmutar de un proceso a otro. Los contenidos actuales de los registros se colocan en una palabra de estado del proceso *B*; los registros son recargados con los valores de la palabra de estado del proceso *A*.

signal. Si el valor del contador es negativo, cualquier proceso que ejecute una instrucción *wait* pasa a la cola; cuando se recibe la siguiente instrucción *signal*, el primer proceso de la cola se transfiere a la "lista disponible", que tiene los procesos preparados para su ejecución. Las dos operaciones definidas sobre los semáforos son lo bastante potentes para sincronizar procesos paralelos en gran variedad de contextos, desde la necesidad de parar un proceso cuando una memoria intermedia de entrada está vacía o una de salida está llena, hasta la necesidad de permitir que un solo proceso manipule datos compartidos.

El nivel 6 de la jerarquía de un sistema operativo gestiona el acceso a los dispositivos de almacenamiento secundario de una máquina particular. A este nivel los programas son responsables de operaciones tales como la ubicación del cabezal de un controlador de disco o la lectura de un bloque de datos. La programación de un nivel más alto determina simplemente la posición de los datos en el disco y coloca una petición para el dispositivo en la cola de trabajos pendientes. El proceso solicitado espera en un semáforo hasta que la transferencia ha concluido.

La función del nivel 7 es la memoria virtual, un mecanismo para gestionar las memorias primarias y secundarias de un computador que da al programador la ilusión de tener una memoria principal lo suficientemente grande para contener un programa y todos sus datos, incluso cuando la capacidad disponible de la memoria principal sea mucho menor. Las direcciones pueden ser arbitrariamente extensas y los programas que se ejecutan concurrentemente pueden emplear la misma dirección sin ningún tipo de conflicto; el sistema operativo traduce cada dirección virtual en una dirección de memoria física. Si falla un intento de traducción porque la información solicitada no está en memoria principal, el gestor de la memoria virtual la busca automáticamente en el disco. Antes de hacer esto, quizá sea preciso dejarle sitio libre en la memoria, quitando para ello otros datos. Como en otras circunstancias similares, el proceso demandante se interrumpe hasta que se obtiene la información necesaria.

Hasta el nivel 7, el sistema operativo se ocupa exclusivamente de los recursos de una sola máquina. A partir del siguiente nivel, los programas del sistema operativo abarcan un mundo más amplio que incluye dispositivos periféricos tales como terminales e impresoras y también otros computadores conectados a la red.

El nivel 8 trata explícitamente de la comunicación entre procesos, que pueden coordinarse a través de un mecanismo de interconexión. Un tubo de interconexión ("pipe") es un canal de sentido único: desde un extremo del tendido fluye una corriente de datos hasta el otro. Dicho flujo tiene un puntero de escritura que indica el número de datos escritos en el canal y un puntero de lectura que registra el número de datos leídos por el otro extremo. Cualquier petición para leer más datos se retrasa hasta que se hallan realmente presentes. El tubo o canal conecta dos procesos que se ejecuten en una única máquina; verbigracia, cuando la salida de un programa es la entrada de otro. El canal transmite correctamente, también, información entre computadores; más aún, un conjunto de canales que conecten procesos de todas las máquinas de una red puede servir de instalación difusora útil para encontrar recursos que pueden estar en cualquier parte de la red.

El sistema de ficheros, que proporciona el almacenamiento permanente de ficheros clasificados, se desarrolla en el nivel 9. Mientras que el nivel 6 se ocupa del almacenamiento en disco en términos de pistas y sectores –las divisiones de tamaño fijo realizadas en los propios circuitos–, el nivel 9 confiere dirección a entidades más abstractas, de longitud variable, cuyos límites no se corresponden necesariamente con los de las pistas y sectores físicos. Un fichero puede estar disperso en muchos sectores no contiguos.

Las operaciones *create* (crear) y *des-*

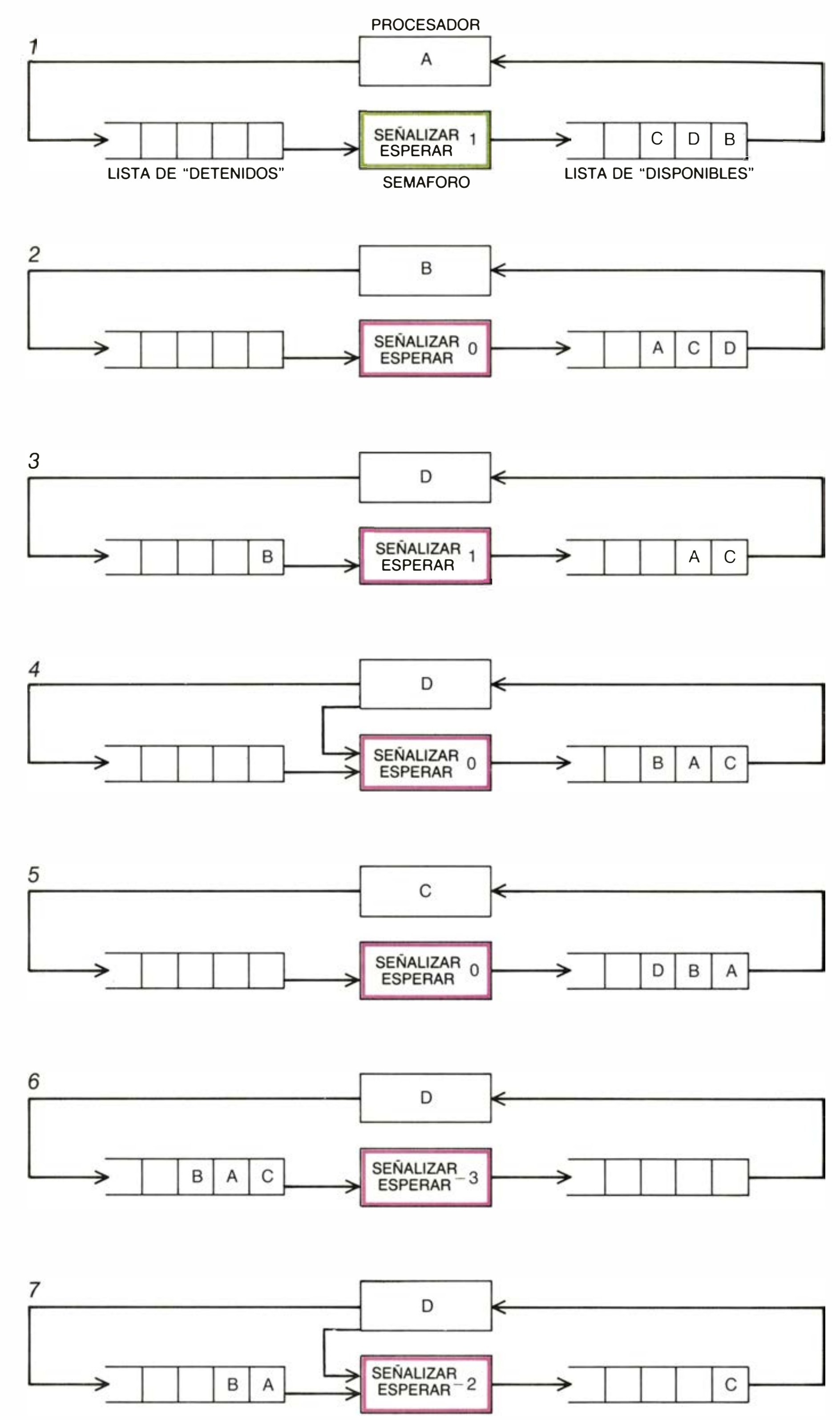
troy (destruir) producen un fichero nuevo y destruyen otro viejo, respectivamente; *open* (abrir) y *close* (cerrar) conectan y desconectan un fichero con un proceso. Para examinar el contenido de un fichero, debe haberse copiado en un área de la memoria virtual; para que se guarde la información, debe copiarse, de la memoria virtual, en el fichero; copia que se realiza a través de las operaciones *read* (leer) y *write* (escribir). Si un fichero está en una máquina diferente, los programas del nivel 9 pueden, usando el nivel 8, lanzar un tendido hasta los ficheros de la máquina donde residen. (Cómo realizar de la mejor manera esta operación constituye todavía un problema abierto.)

El nivel 10 provee el acceso a dispositivos externos de entrada y salida: un reloj que indica el tiempo real, impresoras, trazadores de gráficas y los teclados y pantallas de los terminales. Las operaciones definidas sobre esos objetos son, de nuevo, *create* y *destroy, open* y *close, read* y *write*; también ahora se puede crear un canal para obtener el acceso a un dispositivo conectado a otra máquina.

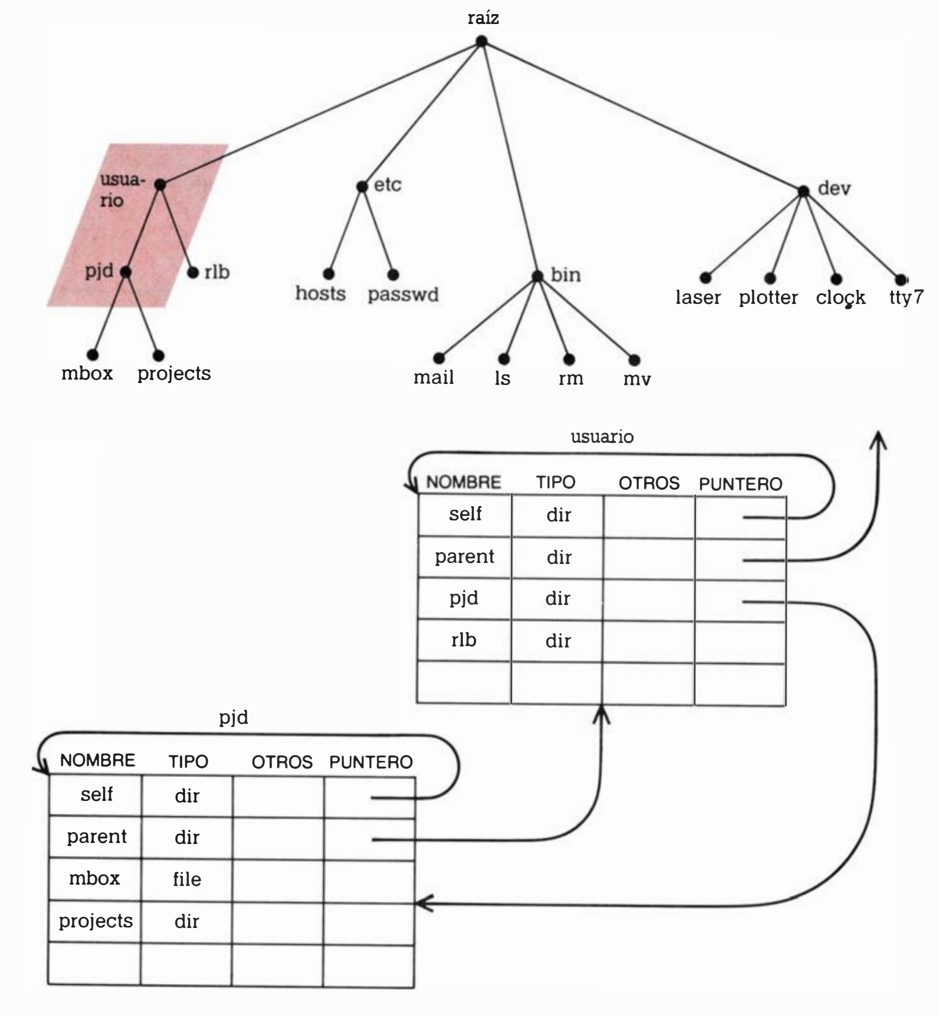
El nivel 11 gestiona una jerarquía de directorios en los que se catalogan los recursos de circuitos y programas cuyo acceso debe estar controlado: canales, ficheros, dispositivos y los mismos directorios. El elemento central de un directorio es una tabla que realiza la correspondencia entre el nombre externo de un objeto (esto es, el nombre conocido y aplicado por el usuario, verbigracia "addresslist") y el nombre interno empleado por el sistema operativo para encontrar el objeto. Es necesaria una jerarquía, puesto que un directorio puede incluir entre sus elementos los nombres de directorios subordinados.

Cada directorio es un catálogo de entradas que proporciona el nombre externo del objeto (almacenado en forma de una cadena de caracteres), el nombre interno (almacenado como un código binario), un indicador de su tipo (fichero, dispositivo, etcétera) y alguna otra información. El directorio revela, normalmente, si el objeto puede o no ser leído o escrito y, en el caso de ficheros de programas, si puede ser ejecutado; un tipo de acceso puede estar permitido para unos usuarios pero no para otros.

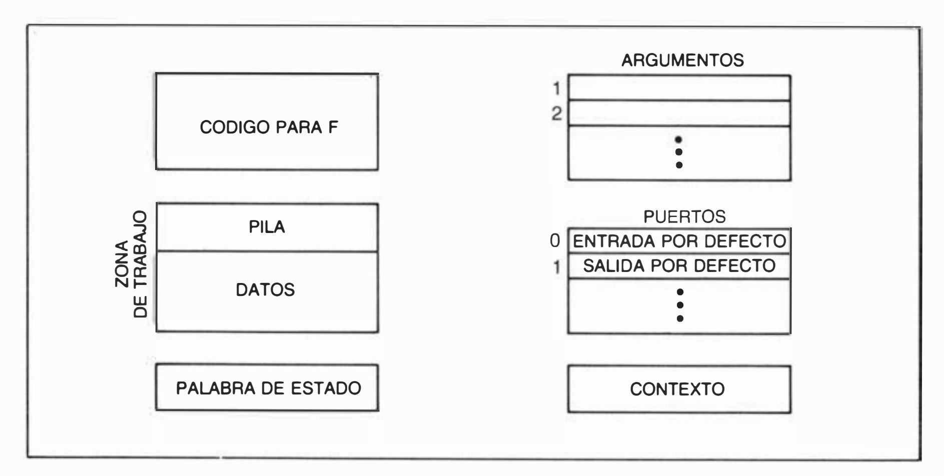
El nivel directorio es responsable sólo de recordar la asociación entre los nombres internos y externos de los objetos; otros niveles gestionan los objetos. Así pues, cuando se busca la secuencia "clock" (reloj) en un direc-



5. SEMAFORO DE PROGRAMACION, mecanismo para controlar procesos primitivos que deben sincronizarse. Aquí, los procesos *A*, *B* y *C* dependen de los resultados del proceso *D*. Por cada instrucción *wait* (aguarde), el semáforo resta un contador y por cada instrucción *signal* lo aumenta. A un proceso que está esperando se le deja pasar el semáforo sólo si el valor del contador es superior a cero. Inicialmente, el proceso *A* se está ejecutando, *B*, *D* y *C* se encuentran preparados y el valor de +1 dibujado en el semáforo indica que está disponible uno de los resultados de *D* (*1*). Cuando *A* ejecuta una instrucción *wait*, pasa inmediatamente el semáforo y llega a la lista de disponibles. Después se ejecuta *B* (*2*), el cual puede llegar a una instrucción *wait* y quedar parado, permitiendo que *D* comience a ejecutarse (*3*). Cuando *D* completa un nuevo resultado, ejecuta una instrucción *signal*, lo que permite llevar a *B* hasta la lista de disponibles (*4*). *D* vuelve a la lista de disponibles; comienza a desarrollarse *C* (*5*), aunque se detiene cuando emite una instrucción *wait*. De la misma forma, *A* y *C* empiezan a ejecutarse y quedan suspendidos, permitiendo que *D* reanude su ejecución (*6*). Cuando *D* tiene un resultado, emite una instrucción *signal*, transfiriendo *C* a la lista de procesos disponibles (*7*). Ciclos posteriores de *D* levantarán la suspensión de *A* y *B*.



6. ARBOL DE FICHEROS Y DIRECTORIOS para organizar los recursos de un computador. La raíz del árbol y los nudos intermedios son directorios que pueden listar ficheros o directorios subordinados. En este sentido, el directorio de "asociación" contiene el código binario de los programas de ayuda (verbigracia, programas para correo electrónico y para listar, transferir o eliminar ficheros). De igual modo, el directorio "dev" tiene una tabla de los dispositivos y el directorio "etc" contiene informaciones diversas, tales como los computadores huéspedes disponibles y los códigos de acceso cifrados de los usuarios. La estructura del directorio "user" (utilizador), donde cada usuario del computador mantiene sus propios ficheros, se muestra con mayor detalle en la parte inferior de la figura. Cada directorio tiene un puntero hacia sí mismo y hacia su predecesor. La estructura representada aquí se basa en la del sistema Unix.



7. UN PROCESO DE USUARIO es un computador virtual: una máquina simulada y dedicada a la ejecución de un programa. Incorpora los elementos del proceso primitivo (el código ejecutable, la zona de trabajo y la palabra de estado), así como una lista de los argumentos suministrados cuando el programa se inició, una lista de los canales de entrada y salida y una descripción del contexto de los programas. Los argumentos son parámetros escritos después del nombre del comando; se incorporan dentro de bloques sucesivos de la cadena ARGS. Los puertos incluyen, por "defecto", a dos de entrada y salida, que sirven de comodín a no ser que se especifiquen otros. El contexto contiene datos tales como el directorio de trabajo.

torio de dispositivos, el resultado que se devuelve al programa solicitante es simplemente el nombre interno del reloj del tiempo actual. El nombre interno se pasa entonces al programa del nivel 10, que realizará la lectura verdadera del reloj.

El nivel 12 acomete los procesos de usuario, verdaderas máquinas virtuales que ejecutan programas. Importa distinguir entre los procesos de usuario y los procesos primitivos de nivel 5. Toda la información necesaria para definir un proceso primitivo puede expresarse en la palabra de estado que almacena el contenido de los registros de la unidad central del proceso. Todo proceso de usuario comprende un proceso primitivo, y bastante más: una memoria virtual que contiene el programa y su zona de trabajo, la información suministrada por el usuario cuando se inició el programa y una lista de otros objetos con los que el proceso puede comunicarse. Los procesos de usuario son mucho más potentes que los primitivos.

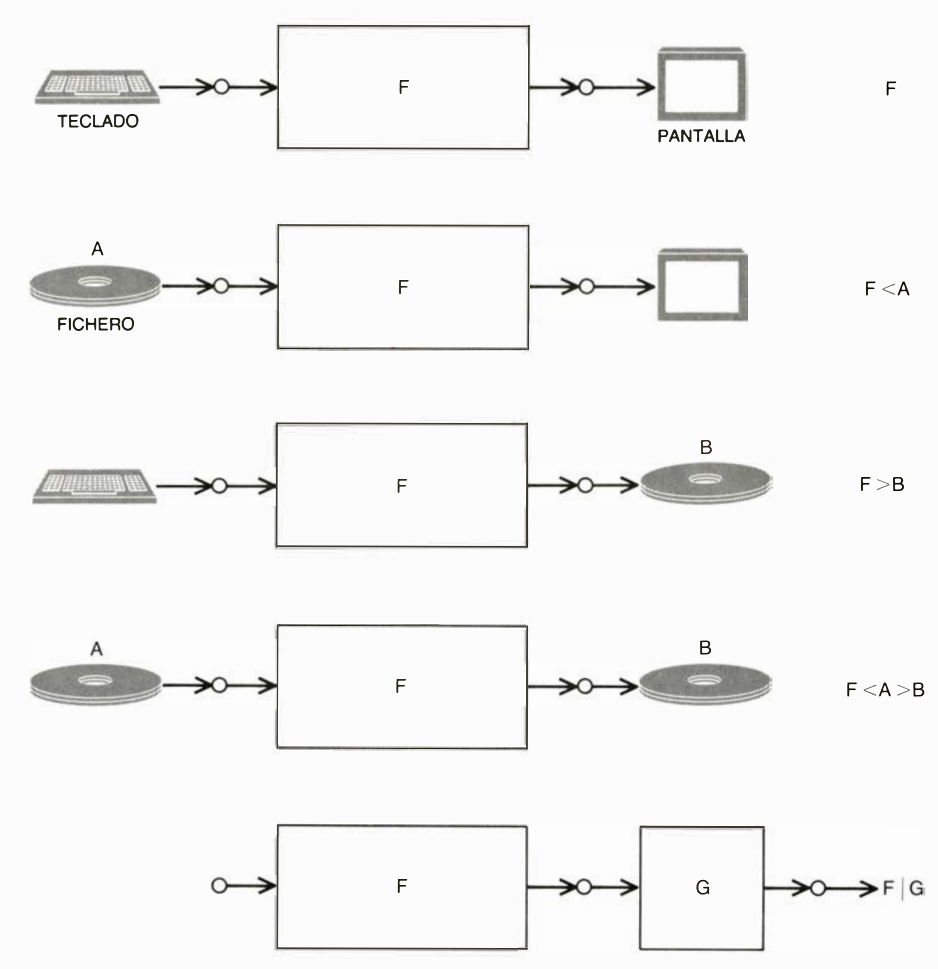
El nivel 13 es el "caparazón" ("shell"), así llamado porque separa al usuario del resto del sistema operativo. Es el intérprete de un lenguaje de comandos, u órdenes, de alto nivel, a través del cual el usuario da instrucciones al sistema. El caparazón incorpora el programa oyente que responde al terminal de teclado; analiza cada línea de la entrada para identificar los nombres de los programas y otras informaciones; crea y llama un proceso de usuario para cada programa y lo conecta a los canales, ficheros y dispositivos necesarios.

Un principio importante adoptado en el hipotético sistema operativo que describimos aquí es el de la independencia de la entrada y salida. En los niveles 8, 9 y 10 se definen las mismas operaciones fundamentales (*create, destroy, open, close, read* y *write*) para tendidos, ficheros y dispositivos. La escritura de un bloque de datos en un fichero del disco requiere una secuencia de sucesos bastante diferentes de los necesarios para transmitir los mismos datos a una impresora o para suministrarlos, como entrada, a otro programa, pero ni el autor ni el usuario del programa necesitan estar al tanto de estas diferencias. Todas las órdenes de *read* y *write* del programa pueden referirse a "puertos" ("ports"), o puertas de entrada y salida. Los puertos se asignan a ficheros, canales y dispositivos particulares sólo cuando se ejecuta el programa.

Esta estrategia, llamada de asociación retardada, puede incrementar enormemente la versatilidad de un programa. Una rutina de clasificación, por ejemplo, puede tomar su entrada desde un fichero o directamente desde un terminal y enviar su salida a otro fichero, terminal o impresora. Sin esta asociación retardada, se necesitaría una rutina distinta para cada combinación posible de una fuente y un destino.

Otro principio seguido al construir un sistema operativo lleva el enigmático nombre de "ocultamiento de la información". Cada nivel hace uso de los niveles inferiores, pero esconde todos los detalles internos de sus operaciones a los niveles superiores. Por ejemplo, el gestor de los procesos básicos del nivel 5 crea la ilusión de que todos los procesos básicos de la lista de disponibles se están ejecutando en paralelo; el guardar cola, las interrupciones, etcétera, son pormenores invisibles para los niveles superiores. Un programa que utilice procesos primitivos maneja sólo un conjunto pequeño de órdenes externas para crear y destruir procesos, suspender y reanudar su ejecución y enviar y recibir mensajes. Similarmente, el gestor de los procesos de usuario en el nivel 12 da la ilusión de que cada programa funciona en su propia máquina; la creación del proceso básico, la zona de trabajo y las conexiones con los canales de entrada y salida permanecen escondidas.

Del mismo modo que ningún nivel tiene acceso a la forma interna de trabajar de los niveles inferiores, tampoco depende de hipótesis realizadas sobre los niveles superiores. El gestor de la memoria virtual (nivel 7) debe tener acceso al sistema de interrupciones (nivel 4) y al sistema de almacenamiento secundario (nivel 6), pero no conoce nada de la estructura de ficheros (nivel 9). La estratificación de un sistema operativo ayuda a su construcción, puesto que los niveles pueden ser instalados y comprobados uno a uno desde el nivel inferior.



8. SE LLAMA COMPONENTE O PARTE DE PROGRAMA al programa dotado de un único flujo de entrada y uno de salida; dicha estructura ayuda a combinar programas y dispositivos de diversas formas. Si no se especifican ni otra fuente ni otro destino, un programa se conecta por ausencia al teclado y la pantalla del usuario. El signo "<" designa una fuente de entrada y el signo ">" un destino para la salida. El signo "|" crea un conducto o canal que une la salida (output) de un programa a la entrada de otro.

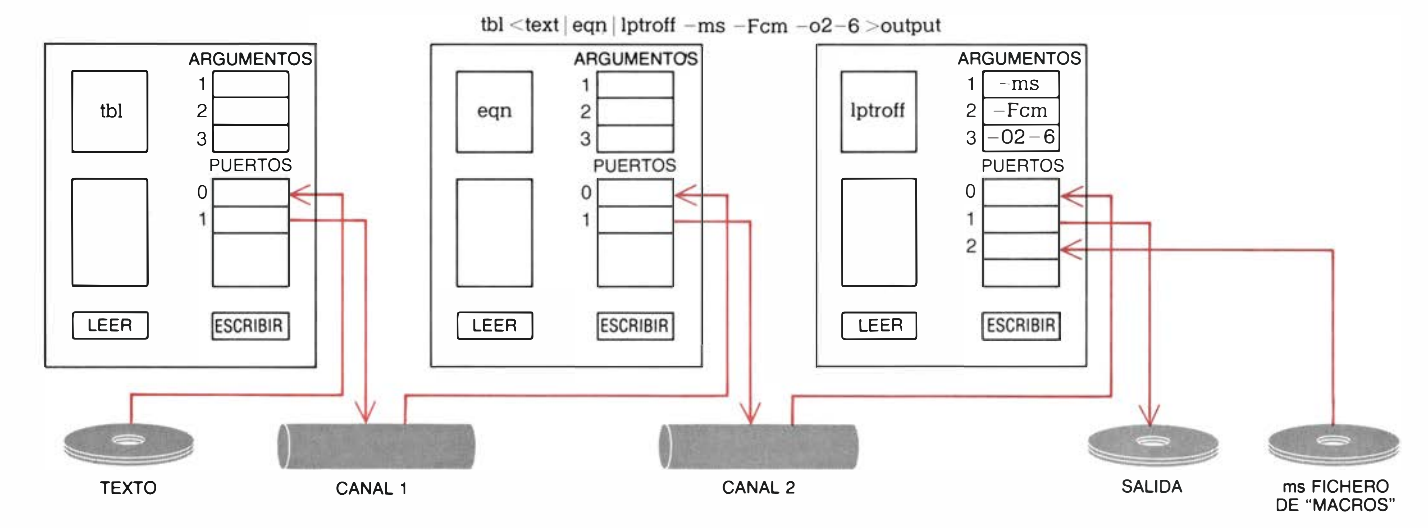
La descripción que hemos realizado, hasta ahora, del sistema operativo es estática: hemos enumerado sus partes, pero no hemos mostrado todavía cómo trabajan. El estado del sistema cambia cuando se van ejecutando los comandos. Los siguientes ejemplos de la dinámica del sistema se basan en el sistema operativo Unix, que incorpora muchas de las características del sistema hipotético examinado hasta ahora.

El usuario ve al computador como un gran sistema dotado de muchos recursos aprovechables; algunos de éstos son programas almacenados en forma de código binario que pueden ejecutarse con sólo dar su nombre de fichero. En un sistema Unix, esta categoría puede abarcar el programa fecha, los compiladores de lenguajes de alto nivel y los programas para preparar documentos, incluidos los encargados de dar formato a tablas, ecuaciones y textos. Otros elementos del sistema son los ficheros de datos portadores quizá de documentos de varias clases, entre ellas el "código fuente" de los programas. También son accesibles dispositivos con circuitos, tales como los terminales, el reloj y las impresoras.

Los directorios que tabulan todos estos recursos están ordenados en un árbol invertido, con el directorio de nivel más alto en la raíz. Algunos directorios están reservados para uso del sistema; entre ellos deben incluirse directorios de dispositivos, de comandos y de ficheros que contienen los códigos de acceso de los usuarios autorizados y otros datos de distinta índole. Un directorio etiquetado "usuario" posee un subdirectorio para cada persona con crédito en el sistema; a su vez, cada usuario puede crear un árbol de subdirectorios adicionales para almacenar todos los ficheros, canales y dispositivos que ha creado.

La mayoría de los usuarios de un sistema consumen más tiempo empleando programas ya existentes que escribiendo otros nuevos. El diseño del sistema operativo y, en particular, el principio de independencia de entrada y salida anima este estilo de computación. Muchos de los programas de las bibliotecas del sistema son "partes logicales" que pueden trabajar en intercambio, con distintas fuentes de entrada y varios destinos para la salida.

Cuando un usuario "entra" en el sistema, a través de un código de acceso, el sistema operativo crea un proceso de usuario, que incluye una copia del programa caparazón. La entrada al capara-



9. PROCESO CONDUCTOR que conecta entre sí tres programas, dos canales o conductos y tres ficheros, para preparar la impresión de un texto. El primer programa, *tbl*, toma su entrada del fichero con nombre "text" e inserta comandos para dar formato en modo tabular. La salida de *tbl* se conduce a *eqn*, que proporciona formatos similares a ecuaciones. El segundo conducto lleva el texto a *lptroff*, que completa el formato; el nombre del programa indica "laser-printer typesetter runoff" (salida por impresora láser). En el comando de línea se ofrecen tres opciones a *lptroff* como argumentos: *-ms* ordena al programa que abra un fichero "macro", llamado "*ms*", a fin de expandir códigos de formato abreviados que se encuentran en el texto, *-Fcm* especifica un tipo de letra denominado Computador Moderno y *-02-6* señala que sólo deben generarse las páginas de salida desde la 2 hasta la 6. La salida se orienta hasta un fichero para su impresión, que se realizará en una impresora láser, máquina de alta calidad que trabaja como una fotocopiadora.

zón está conectada al teclado del terminal del usuario y la salida a la pantalla del mismo terminal. El caparazón "escucha", sin tomar acción alguna, hasta que se ha tecleado una línea entera, señalizada por la recepción del carácter de retorno. Se explora entonces la línea para extraer los nombres de los programas solicitados y los valores que han de pasarse como argumentos para dichos programas. Por cada programa que se llame mediante este mecanismo, el caparazón crea un proceso de usuario, que contiene una copia del código ejecutable para el programa y una zona de trabajo. Los procesos se ponen en conexión de acuerdo con el flujo de datos especificado en los comandos de la línea.

Con el lenguaje de comandos del caparazón Unix pueden especificarse operaciones de notable complejidad. Por ejemplo, una secuencia de operaciones que da formato a un fichero denominado "texto" puede ponerse en marcha mediante la línea de comandos

tbl < texto | eqn | lptroff > output.

Aquí, el primer programa en ser llamado es *tbl*, cuya función consiste en buscar en un fichero las descripciones de tablas de información e insertar los comandos para dar formato. El símbolo "<" indica que *tbl* debe tomar su entrada desde el fichero "texto". La salida de *tbl* se dirige por un conducto (el símbolo "|") hacia la entrada de *eqn*, que suministra comandos para dar formato a cualquier descripción de las ecuaciones. La salida de *eqn* se lleva entonces por un canal a *lptroff*, que es otro programa para dar formatos que prepara el resto del texto para imprimirlo; el nombre del programa es la abreviatura de "laser printer typesetter runoff" (salida por impresora láser). Finalmente, el símbolo ">" designa que el documento con su formato se escribe en un fichero denominado "output". Este último está preparado para enviarse a una impresora láser, una impresora de alta calidad que trabaja de manera parecida a una fotocopiadora.

Si se ha de realizar frecuentemente las operaciones de dar formato e imprimir documentos, escribir un comando tan largo sería demasiado tedioso. Unix anima al usuario a almacenar comandos complicados en ficheros ejecutables, denominados descriptores, los cuales se convierten en comandos más simples. Para ello, se debe crear un fichero *lp* con el siguiente contenido

tbl < $1 | eqn | lptroff > $2

Se han sustituido los nombres de los ficheros de entrada y salida por variables $ 1 y $ 2. Cuando se llama al comando *lp*, las variables son remplazadas por los argumentos que siguen al comando. Por ejemplo, escribiendo

lp texto output

se sustituiría "texto" en $ 1 y "output" en $ 2, con lo cual tendríamos el mismo efecto que con el comando más largo escrito anteriormente.

Nos resta una parte menor, aunque esencial, del sistema operativo. Dado que varias componentes del sistema tienen la responsabilidad de cargar todos los programas en la máquina, la cuestión que se nos plantea es cómo se carga el propio sistema operativo y cómo se empieza a ejecutar. La contestación está en una "secuencia de autoarranque", que comienza con un programa de dos instrucciones permanentemente escritas en memoria de sólo lectura y, por tanto, presentes siempre, aun cuando el computador quede desconectado de la red eléctrica. Este pequeño programa inicia la carga desde el disco de otro programa más grande que, entonces, toma el control y carga al propio sistema operativo.

El principio de jerarquía que hemos aplicado aquí a la organización de un sistema operativo ha demostrado ser de gran utilidad en las ciencias naturales. Después de todo, las estructuras y los sucesos del mundo real abarcan muchos órdenes de magnitud en el espacio y en el tiempo, por lo que no pueden tenerse en cuenta todos a la vez; no es posible comprender en su totalidad la evolución de una galaxia mediante el trazado de las trayectorias de sus átomos constituyentes. De todos los objetos hechos por el hombre, los computadores tienen la disparidad mayor entre los componentes menores y los grandes. Los diseñadores de sistemas operativos han comenzado a cubrir este amplísimo margen de escalas creando una jerarquía de abstracciones.

Programación y tratamiento de lenguajes

Los programas manipulan símbolos lingüísticos con gran facilidad, como en el tratamiento de textos, pero los esfuerzos por conseguir que los computadores manejen el significado se ven frustrados por la ambigüedad de los idiomas

Terry Winograd

En la mitología popular, el computador es una máquina matemática: está diseñada para realizar cálculos numéricos. Pero lo cierto es que se trata de una máquina lingüística: su poder fundamental reside en su capacidad de manipular símbolos lingüísticos, a los que se ha asignado un significado. El propio "lenguaje natural" (el que la gente habla o escribe, en oposición a los "lenguajes artificiales", en los que se escriben los programas de ordenador) es parte central de la informática. Los primeros trabajos empeñados en este campo se proponían, sobre todo, descifrar los códigos militares; en la década de los 50 los esfuerzos por lograr que los ordenadores tradujeran un texto de un lenguaje natural a otro cristalizaron en avances importantísimos, aunque no se alcanzó la meta prefijada. El trabajo prosigue en un proyecto aún más ambicioso: convertir el lenguaje natural en un medio de comunicación con los ordenadores.

Hoy, los investigadores están desarrollando teorías unificadas de la computación que abarcan ambos lenguajes, naturales y artificiales. Me centraré en los primeros, es decir, el lenguaje de la comunicación cotidiana. Dentro de este campo hay una gama enorme de soporte lógico a considerar. Parte de él es trivial y goza de gran éxito. Un aluvión de microordenadores ha invadido los hogares, las oficinas y las escuelas; la mayoría de los cuales se usan, en parte al menos, para procesar textos. Otras aplicaciones, especulativas, están muy lejos de llevarse a la práctica. La fantasía científica llena sus creaciones de robots que hablan como humanos, sin más que un ligero timbre mecánico en su voz. Cuantos intentos se han desarrollado para fabricar computadores que hablen se han topado con dificultades enormes, y a los mejores prototipos de laboratorio todavía les falta mucho por alcanzar la capacidad lingüística media de un niño.

La extensa gama de programación disponible para procesar lenguajes imposibilita un estudio detallado. Ceñiré mi examen a cuatro tipos de programas; a saber: los que se ocupan de la traducción mecánica, del tratamiento de textos, de responder preguntas en un diálogo y de las extensiones del correo electrónico que se conocen por sistemas de coordinación. En cada caso, la clave de lo posible yace en el análisis de la competencia lingüística y en cómo dicha competencia está relacionada con las estructuras formales normativas que constituyen la base teórica de todo soporte lógico.

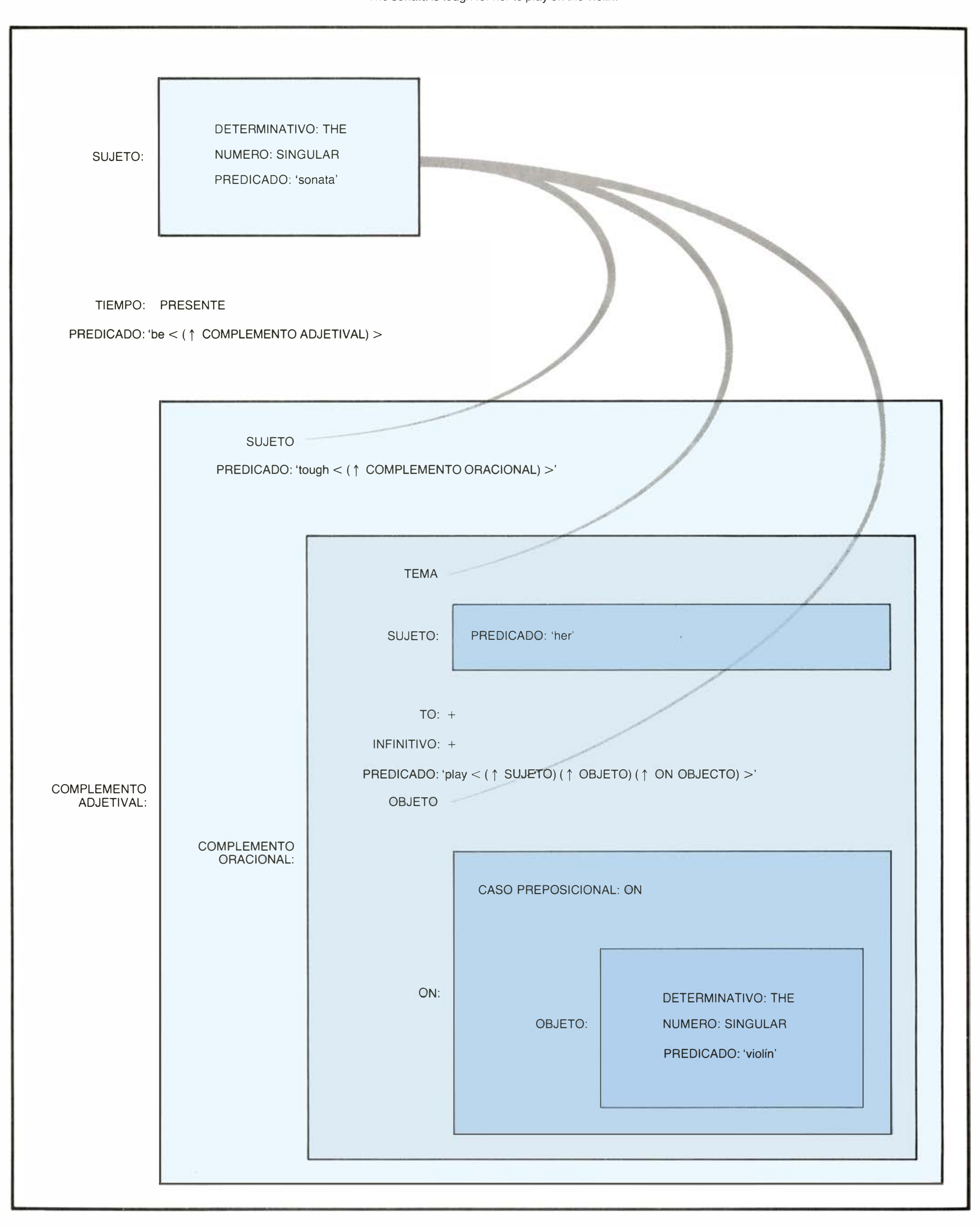
La posibilidad de la traducción mecánica por un ordenador surgió mucho antes de la fabricación comercial de éste. En 1949, cuando los contados ordenadores que estaban en funcionamiento se encontraban en instalaciones militares, Warren Weaver, un matemático pionero de la teoría de la comunicación, señaló que las técnicas desarrolladas para descifrar códigos podrían aplicarse a la traducción mecánica.

En principio, la tarea parece sencilla. Dada una oración en un lenguaje fuente, dos operaciones básicas proporcionan la correspondiente en el lenguaje destinatario. Primero se sustituyen las palabras componentes por su traducción; a continuación, se reordenan las palabras de la traducción y se ajustan los detalles. Sea la traducción de "Did you see a white cow?" al castellano "*¿Viste una vaca blanca?*" Hay que conocer primero las correspondencias de los términos: "*vaca*" con "cow", etcétera. Luego necesitamos conocer la estructura del castellano. Las palabras "did" y "see" no se traducen directamente; se expresan con la forma del verbo "*viste*". El adjetivo "*blanca*" sigue al nombre, no lo antecede como ocurre en inglés. Por último, "*una*" y "*blanca*" están en la forma femenina concordando con "*vaca*". Las primeras investigaciones en traducción mecánica insistían, sobre todo, en el problema técnico que comportaba la introducción de un diccionario grande en la memoria del computador y el dotarla de medios para buscar eficazmente en él. Al propio tiempo, la programación relativa al manejo de la gramática se basaba en las teorías entonces al uso sobre la estructura del lenguaje, más algunas reglas improvisadas.

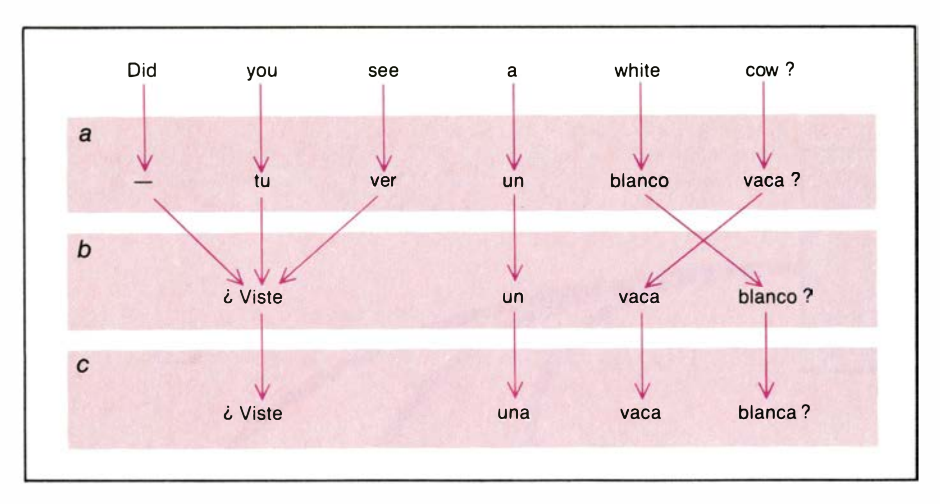
Los programas daban unas traducciones tan malas que resultaban ininteligibles. El problema estribaba en que los lenguajes naturales expresan el significado de una manera distinta de como los códigos criptográficos expresan los mensajes. El significado de una frase en el lenguaje natural no depende sólo de la forma de la oración, sino también de su contexto. Algunos ejemplos de ambigüedad nos lo van a aclarar en seguida.

En su forma más sencilla, la llamada ambigüedad léxica, una palabra tiene más de un sentido posible. Así "Stay away from the bank" (N. del T. "*No te acerques al banco" o "No te acerques a la orilla*") puede ser un consejo a un inversionista o a un niño que está demasiado cerca de un río. Traduciéndolo al castellano, hay que elegir entre "orilla" y "banco"; nada hay en la frase misma que nos revele el significado pretendido. A la hora de intentar solucionar la ambigüedad léxica, en la programación de la traducción, se ha atendido a la in-

The sonata is tough for her to play on the violin.



1. REPRESENTACION DE UNA ORACION para poner de manifiesto las relaciones lingüísticas entre sus partes; tal ha sido el propósito de la ciencia lingüística. Pero constituye también un aspecto esencial del trabajo implicado en la programación de ordenadores que "comprendan" el lenguaje, o por lo menos saquen inferencias de las entradas lingüísticas. Se ilustra aquí una oración en forma de "estructura funcional", que tiene la propiedad de que cuando una parte de una oración desempeña una función en otra parte, la primera está "anidada" en la segunda. Esa inclusión se demuestra introduciendo un recuadro en el interior de otro, o (en tres lugares) por una "cadena". La frase –"The sonata is tough for her to play on the violin"– fue analizada por Ronald M. Kaplan y Joan Bresnan, de la Universidad de Stanford y del Centro de Investigaciones de Palo Alto de la Compañía Xerox. Otro diagrama de estructura funcional aparece con todo detalle en la figura 10. La oración podría traducirse así: "Es muy dura la sonata para que la interprete al violín".



2. TRADUCCION MECANICA de un texto de un lenguaje a otro. En los años cincuenta, cuando comenzó este tipo de trabajo, se confiaba en su viabilidad. Como primer paso del proceso (*a*), el ordenador buscaría en un diccionario bilingüe hasta dar con la traducción de las distintas palabras que componían la frase fuente (en este caso, las equivalentes españolas de las palabras de la frase "Did you see a white cow?"). Después, se cambiaría el orden de las palabras traducidas según la gramática del idioma al cual se vertirían (*b*). Los cambios en este momento podrían incluir la eliminación o la incorporación de palabras. Por fin, se modificaría la morfología de la traducción (por ejemplo, los morfemas de género) (*c*).

serción, en el texto traducido, de todas las alternativas, así como al análisis estadístico del texto original, para decidir qué traducción es la apropiada. Por ejemplo, "orilla" será, probablemente, la correcta si en el texto fuente se encuentran cerca palabras relacionadas con el agua y los ríos. La primera estrategia nos ofrece un texto complejo e ilegible; la segunda elige bien en muchos casos, pero mal en muchos otros.

En la ambigüedad estructural, el problema trasciende el mero término. Consideremos la frase "He saw that gasoline can explode". Tiene dos interpretaciones basadas en los usos totalmente distintos de "that" y de "can". (N. del T. "*El vio que la gasolina puede explotar*" o "*El vio explotar aquella lata de gasolina*".) Así, la frase tiene dos estructuras gramaticales posibles; el traductor debe elegir entre ellas.

Más sutil todavía se presenta la ambigüedad de "estructura profunda": dos lecturas de una frase pueden tener la misma estructura gramatical aparente y sin embargo diferir en su significado. "The chicken are ready to eat" implica que algo está a punto de comer algo; pero, ¿los pollos comerán o serán comidos? (N. del T. "*Los pollos están listos para comer*" o "*Los pollos están a punto de comer*".) Uno de los avances en la teoría lingüística desde los años cincuenta ha sido el desarrollo de un formalismo que puede representar la estructura profunda del lenguaje, pero este formalismo ayuda poco a determinar la estructura profunda deseada de una frase concreta.

Vayamos con el cuarto tipo de ambigüedad: la ambigüedad semántica. Se presenta cuando una frase puede tener distintas funciones en el sentido global de una oración. La frase "David wants to marry a Norwegian" (N. del T. "*David quiere casarse con una noruega*") es un ejemplo. En uno de los sentidos de la oración, la expresión "a Norwegian" es referencial. David muestra la intención de casarse con una persona concreta, y el hablante ha escogido un atributo de la misma –su nacionalidad– para describirla. En otro sentido no, la expresión es atributiva. Ni David ni el hablante piensan en una persona concreta; la oración significa simplemente que David quiere casarse con una mujer de nacionalidad noruega.

Un quinto tipo de ambigüedad puede llamarse "ambigüedad pragmática". Surge del uso de los pronombres y de palabras especiales como "one" y "another" (otro/a). Sea la oración "When a bright moon ends a dark day, a brighter one will follow" (N. del T. "*Cuando una luna brillante pone fin a un día oscuro, le/la seguirá uno/a más brillante*"). ¿Un día más brillante o una luna más brillante? A veces el programa de traducción puede traducir el pronombre o nombre ambiguo, manteniendo así la ambigüedad en la traducción. En muchos casos no se puede emplear esta estrategia. En la traducción al castellano de "She dropped a plate on the table and broke it" (N. del T. "*Ella dejó caer un plato sobre la mesa y lo/la rompió*"), hay que elegir entre el masculino "*lo*" y el femenino "*la*" para traducir "it". La elección obliga al traductor a decidir si se rompió el *plato* (masculino) o la *mesa* (femenino).

En muchas frases ambiguas el sentido es obvio para un lector humano, pero sólo porque él aporta su comprensión del contexto. Así "The porridge is ready to eat" (N. del T. "*La papilla está lista para comer*") no es ambigua porque uno sabe que la papilla es inanimada. "There is a man in the room with a green hat on" (N. del T. "*Hay un hombre en la sala que lleva un sombrero verde*") tampoco es ambigua, porque uno sabe que las salas no llevan sombrero. Sin esos conocimientos cualquier frase resultaría ambigua.

Aunque no es posible una traducción mecánica de alta calidad y totalmente automática, existe soporte lógico que facilita la traducción. Tenemos un ejemplo de ello en la informatización de medios auxiliares: diccionarios y libros de frases, que van desde los sistemas complejos para los traductores técnicos, en los que la función de buscar una palabra es parte de un programa multilingüe de tratamientos de textos, hasta las "bibliotecas" de frases de bolsillo para los turistas. También cabe otra estrategia: procesar a mano el texto para adecuarlo a la traducción mecánica. Piénsese en un redactor que, ante un texto del lenguaje original, reescribe un segundo texto, en el mismo lenguaje original, simplificado ahora de suerte que haga más fácil la traducción mecánica. Se eliminan las palabras de múltiple significado y las construcciones gramaticales que complican el análisis sintáctico. Se suprimen las conjunciones que originan ambigüedades o se resuelven éstas añadiendo una puntuación especial, por ejemplo "the [old men] and [women]", ("Los [viejos] y [mujeres].) Después de la traducción mecánica, otro redactor limpia de errores el texto traducido y lo pule.

El esfuerzo resulta a veces rentable. Ni el primer redactor ni el segundo necesitan ser bilingües, a diferencia del traductor. Además, si un texto (por ejemplo un manual de instrucciones) ha de traducirse a varios idiomas, quedará justificada una inversión grande en la primera redacción porque servirá para todas las traducciones. Si el autor del texto domina la forma menos ambigua del lenguaje fuente, sobrará el trabajo del primer redactor. Por último, la programación puede ayudar a revisar el texto preparado en la primera redacción para asegurarse de que cumple las es-

pecificaciones que requiere su introducción en el sistema de traducción (aunque esto no garantiza que la traducción sea aceptable).

Desde 1980, la Organización Panamericana de la Salud viene usando un sistema de traducción mecánica que emplea la doble redacción: anterior y posterior a la traducción; ha traducido más de un millón de palabras del español al inglés. Se está desarrollando un nuevo sistema para la Comunidad Económica Europea que traduzca documentos entre los idiomas oficiales de la comunidad: alemán, danés, francés, holandés, inglés e italiano. Mientras tanto, prosigue el trabajo teórico sobre la sintaxis y el significado, pero no se ha conseguido ningún adelanto decisivo en el campo de la traducción por ordenador. La ambigüedad que atraviesa el lenguaje natural sigue limitando las posibilidades por razones que examinaré más adelante.

Vayamos al tratamiento de textos, es decir, a la programación que ayuda a preparar, dar formato e imprimir un texto. Los procesadores de textos sólo se encargan del manejo y la puesta en pantalla de cadenas de caracteres; por tanto, sólo se relacionan con los aspectos superficiales de la estructura del lenguaje. Aun así, plantean problemas técnicos de central importancia en el diseño de la programación. En algunos casos, el producto final de un programa de tratamiento de textos es una mera secuencia de líneas de texto. En otros, es un despliegue complejo de elementos tipográficos, a veces con dibujos intercalados. En otros, por fin, constituye un documento estructurado con títulos de capítulos, números de secciones, etcétera, amén de índices general y de materias, compilados por el programa.

Los problemas fundamentales relativos al diseño de programación para el tratamiento de textos radican en la representación y en la interacción. Por representación se entiende la construcción de estructuras de datos que el programa pueda manipular convenientemente, teniendo en cuenta además los temas que le interesan al usuario del sistema, por ejemplo, la maqueta de la página impresa. La interacción se refiere a cómo el usuario expresa sus instrucciones y cómo le responde el sistema.

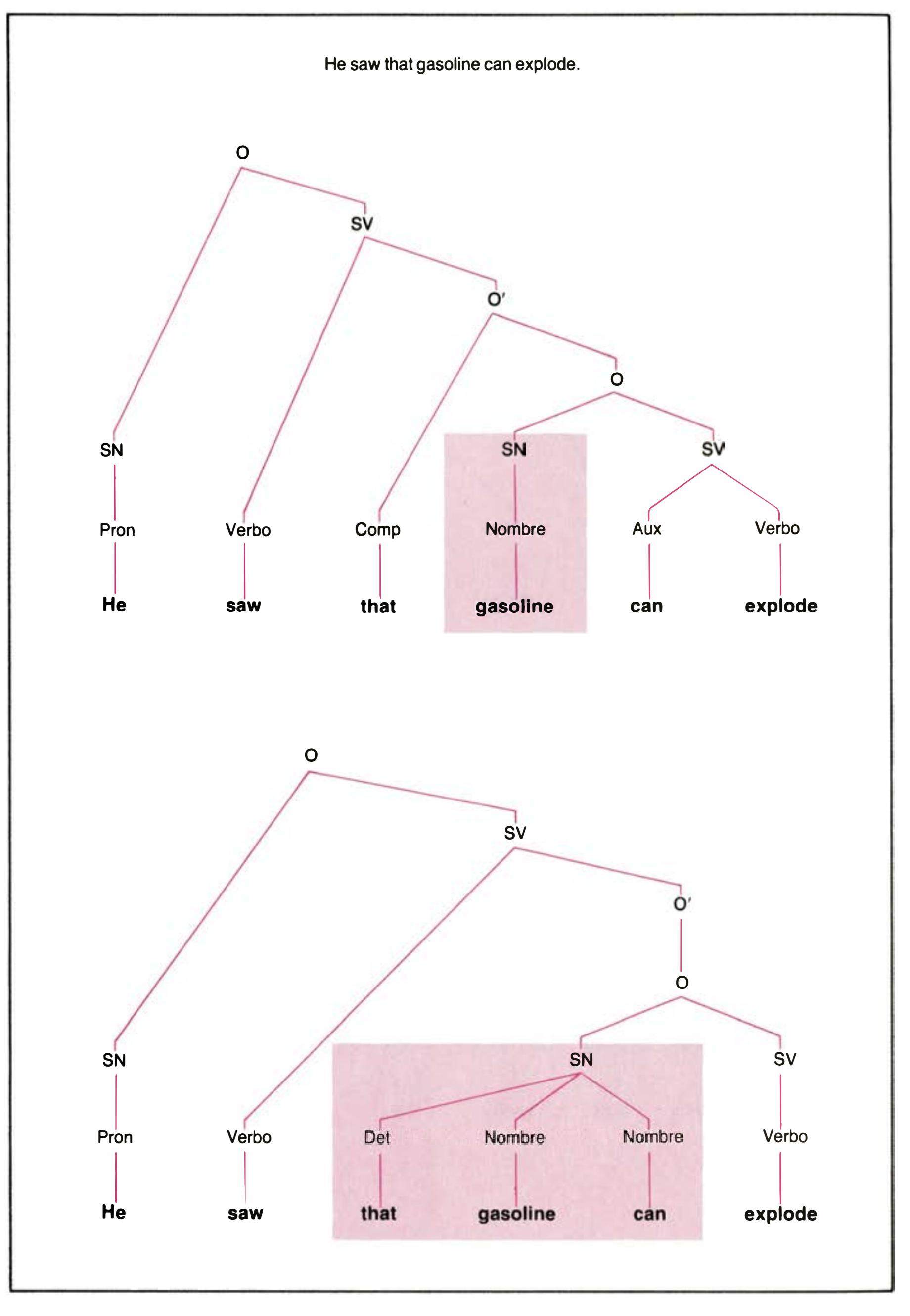
Consideremos un problema fundamental: el empleo de los dispositivos de memoria de un ordenador para almacenar una secuencia codificada de carac-

Stay away from the bank.

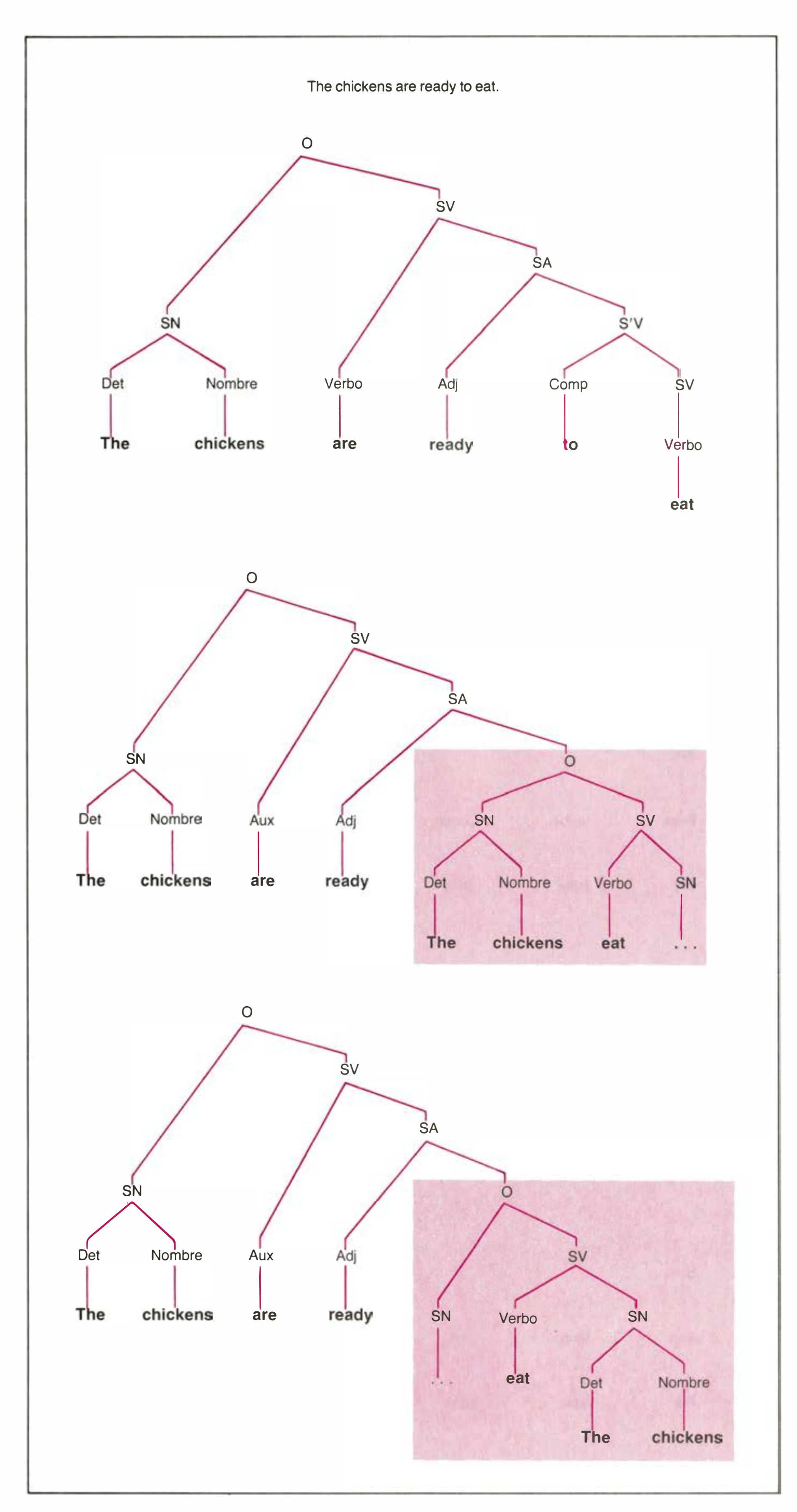
bank *n* 1. the rising terrain that borders a river or lake.

bank *n* 2. an establishment for the deposit, loan, issuance and transmission of money.

3. DE SIGNIFICADOS AMBIGUOS están impregnados los lenguajes naturales (es decir, los idiomas que la gente habla y escribe). Contrarrestan así cuantos esfuerzos se acometen por viabilizar la traducción mecánica de textos de un idioma a otro. El diagrama ilustra el tipo más sencillo, la ambigüedad léxica encerrada en la oración: "Stay away from the bank". En esta ambigüedad una palabra de una frase tiene más de un posible significado. La palabra es "bank" (*color*), que podría referirse o a un río o a una institución financiera. Se debe elegir entre "No te acerques a la orilla" y "No te acerques al banco".



4. SURGE LA AMBIGUEDAD ESTRUCTURAL cuando una oración puede describirse por más de una estructura gramatical. Aquí se exponen las posibilidades conflictivas de la frase "He saw that gasoline can explode" en la forma de "árboles" gramaticales. En el árbol de arriba la frase completa tiene una oración subordinada, cuyo sujeto (*color*) es "gasoline" (gasolina). La oración se refiere al reconocimiento de una propiedad de esa sustancia. La traducción sería: "Vio que la gasolina puede explotar". En el otro árbol "gasoline can" es parte de un sintagma nominal (SN) y significa un contenedor de gasolina; la frase completa se refiere a la observación de una explosión específica: "Vio explotar aquella lata de gasolina".



5. ESTAMOS ANTE UNA AMBIGUEDAD DE ESTRUCTURA PROFUNDA cuando una oración tiene una sola estructura aparente, aunque encierra más de un significado. La oración a analizar es "The chickens are ready to eat": "Los pollos están listos para comer". Su estructura gramatical (*arriba*) deja ambiguo el papel de los pollos: en una interpretación comerán, en la otra serían comidos. Los árboles de estructura profunda sacan a luz el papel de los pollos: son el sujeto de la oración (*centro*), en cuyo caso su comida no está determinada, o son el objeto (*abajo*) y sus comedores están indeterminados.

teres del lenguaje natural. Los primeros dispositivos para codificar textos fueron las tarjetas perforadas y los teletipos; los esquemas de codificación de textos se ajustaban, pues, a esos dispositivos. El teletipo es una especie de máquina de escribir que convierte una pulsación de tecla en un código numérico que se puede transmitir electrónicamente. Así, hay códigos de teletipo para la mayoría de las teclas de una máquina de escribir; aquellos comprenden los caracteres del alfabeto desde la *A* hasta la *Z*, los dígitos de 0 a 9 y los signos de puntuación usuales, como el punto y la coma. Más difícil resulta, sin embargo, establecer normas para ciertos símbolos como #, @, ¢ y}; por no mencionar las teclas que no imprimen nada, como la tecla de tabular, la palanca de retorno de carro y la tecla para retroceder un espacio.

De las dificultades que surgen al elegir una norma nos da idea una peculiaridad de la codificación del texto. El código del teletipo distingue entre el retorno del carro (que devuelve el carro al principio de la línea sin avanzar el papel) y el avance del papel (que no cambia la posición del carro). Por tanto, el final de una línea se señala con una secuencia de dos caracteres: la vuelta del carro y el avance del papel. Bastaría un código; y así, algunos programas eliminan la vuelta del carro o el avance del papel, si no reemplazan ambos caracteres por un código totalmente nuevo. El problema se plantea cuando los programas emplean distintas convenciones, de suerte que las líneas codificadas por uno resultan ilegibles para otro.

Los problemas se enredan en cuanto consideramos una gama completa de caracteres: signos de puntuación, símbolos matemáticos y signos diacríticos, como la diéresis. Por si fuera poco, el tratamiento de textos se está abriendo al chino y al japonés, idiomas que requieren miles de caracteres ideográficos, así como al hebreo y al árabe, idiomas que se escriben de derecha a izquierda. Los esquemas de códigos que son adecuados para el inglés se muestran inútiles para alfabetos con miles de caracteres. Recuérdese, además, que los planteamientos siguen variando, porque las fuerzas económicas y políticas influyen en el diseño de sistemas de ordenadores. Cada fabricante pretende promulgar normas que favorezcan sus propios equipos; normas existen hoy que se deben al vendedor que domina el mercado. Por otra parte, aspectos técnicos como el rendimiento de cierto

soporte lógico ejecutado en determinados equipos perpetúan las diferencias de detalle. Aún falta mucho para que aparezcan normas universales que permitan a los usuarios disfrutar de la posibilidad de transferir textos de un sistema de tratamiento a otro.

Amén de los sistemas de codificación, está la propia forma de las letras. En el teclado de una máquina de escribir una A es simplemente una A. Desde el punto de vista tipográfico, sin embargo, una A es una A, una *A* o una **A**... En el nuevo campo de la tipografía digital, la computadora es una herramienta para diseñar y presentar formas tipográficas. Algunos de los esfuerzos empeñados en este campo se han dedicado a las formas mismas: especialmente, la representación de caracteres como conjuntos de puntos y espacios. Otros esfuerzos van dirigidos a inventar un código para almacenar en computadoras textos que contienen distintos diseños (como Times Roman y Helvética) y distintos tipos (como *cursiva* o **negrita**).

David wants to marry a Norwegian.

∃x Norwegian(x) ∧ Want(David,[Marry(David,x)])

Want(David,[∃x Norwegian(x) ∧ Marry(David,x)])

6. SURGE LA AMBIGUEDAD SEMANTICA cuando una frase puede desempeñar distintos papeles en el sentido de una oración. Aquí se pone de manifiesto qué designa la frase "a Norwegian" cuando la frase "David wants to marry a Norwegian" ("David quiere casarse con una noruega") se traduce en forma lógica, en la notación denominada "cálculo de predicados". Según una interpretación, el hablante está pensando en una persona particular y ha decidido referirse a la nacionalidad de la misma para especificar quién es. De aquí que la frase significa: existe (∃) un *x*, tal que *x* es noruega y (∧) *x* es la persona con quien David quiere casarse. Según la otra interpretación, ni David ni el hablante están pensando en nadie en concreto. Quizá David se vaya a Noruega esperando conocer a una mujer con quien casarse.

She dropped the plate on the table and broke it.

She dropped the plate on the table and broke [the plate].

She dropped the plate on the table and broke [the table].

7. SE PRESENTA LA AMBIGUEDAD PRAGMATICA cuando una palabra, tal el pronombre "it", confiere a la oración más de un posible significado. Supóngase que se dé al ordenador la frase superior: "Se le cayó el plato sobre la mesa y se rompió". Si el ordenador tiene acceso a los conocimientos de la gramática de frases inglesas, pero carece de acceso al conocimiento directo de las propiedades de las mesas y los platos, podría deducir, y con validez, que se rompió la mesa o que lo hizo el plato.

Hasta ahora sólo he hablado de secuencias de caracteres almacenados. Pero una de las principales misiones de un programa de tratamiento de textos es la de controlar los márgenes y el espaciado –la "geografía" de la página impresa–. En el lenguaje TEX de composición tipográfica, las órdenes que especifican los caracteres no habituales cambian el diseño del tipo, ponen los márgenes, etcétera, se hallan insertas en el texto [*véase la figura 8*]. Una orden para TEX se distingue del texto ordinario por una "barra inclinada" (\). El programa TEX "compila" el texto almacenado e interpreta las órdenes insertas para crear un documento impreso con el formato especificado.

La compilación es bastante compleja. A menudo se necesita mucha labor de computación para obtener, de un código creado por un programa de tratamiento de textos, otro que esté listo para atacar una impresora o una máquina de composición. Un algoritmo para ajustar un texto (llenar toda la extensión de cada línea) debe determinar cuántas palabras cabrán en una línea, cuánto espacio quedará entre las palabras y si se puede mejorar una línea dividiendo una palabra mediante un guión. El algoritmo puede intervenir también para evitar fallos estéticos: piénsese en una línea con grandes intervalos vacíos entre palabras seguida de otra apretada y muy compacta. La distribución de cada línea en la página se complica aún más con la ubicación de los titulares, notas al pie, figuras, tablas, etcétera. Las fórmulas matemáticas tienen sus propias reglas tipográficas.

TEX y otros programas parecidos resultan primitivos en comparación con otro aspecto del procesador de textos: la comunicación interactiva (interfaz) con el usuario. Las pantallas de alta resolución ahora disponibles posibilitan que el computador ofrezca al usuario una aproximación bastante buena de las páginas a imprimir, incluyendo la ubicación de cada elemento y el tipo que se va a usar. El usuario no necesita, pues, teclear secuencias especiales de órdenes; le basta con manipular directamente la geografía de la página en la pantalla mediante el teclado del ordenador y un indicador, por ejemplo, un "ratón". La interfaz resultante entre el computador y el usuario sería de la clase conocida por WYSIWYG que significa "What you see is what you get" ("*Lo que vd. ve es lo que vd. obtiene*").

Conviene señalar que lós programas de manipulación de textos reciben nombres distintos en las diferentes profesiones. Los programadores los llaman "editores de texto", pero en el mundo de los negocios y de las editoriales se les llama procesadores de palabras: en estas últimas esferas un redactor es la persona cuyo trabajo consiste en mejorar la calidad de un texto. También se está creando soporte lógico para ayudar en este aspecto más sustantivo de la redacción. No se refiere al formato visual del lenguaje, ni al contenido conceptual, sino a la ortografía, la gramática y el estilo. Abarca dos tipos de programas: trabajos de referencia automatizados y ayudas en la corrección asimismo automatizada.

Un ejemplo de trabajo de referencia automatizado lo tenemos en el programa de sinónimos, cuyo diseño determina que, cuando el escritor señala una palabra, aparezca en pantalla toda una lista de sinónimos. En los programas avanzados, el diccionario de sinónimos está plenamente integrado en el programa de procesamiento de textos. El escritor sitúa un señalizador que apunte la palabra a reemplazar. Se llama, entonces, al diccionario de sinónimos, que muestra las alternativas en una "ventana" en la pantalla. El escritor señala una de las alternativas que a continuación sustituirá automáticamente a la palabra rechazada.

El diseño de un programa así implica nociones de lingüística y de computación. Con la lingüística tiene que ver el mecanismo de búsqueda de palabras que sería suficientemente flexible para aceptar distintas formas. Por ejemplo, los datos referentes a "posibilitar" deben ser asequibles a las consultas acerca de "posible", "posibilidad",

a

```
\inset
This is a sample of a {\italic justified} piece of text, which contains {\eightpoint small letters {\bold and }} {bigFont big ones}.
It includes foreign words such as \lquote pe\~na\rquote—which is Spanish—and foreign letters like \alpha\ and \aleph,
which can be baffling, and includes one \hskip 1.3in wide space.
```

b ...

01110100	01100101	01110010	01110011	00000000	00100111	00101101	11010011	00001000	01100001	01101110	01100100	00000000
t	e	r	s	NEW ENTITY	FONT CODE	X-POSITION	Y-POSITION	X-INCREMENT	a	n	d	NEW ENTITY

00110100	00110001	10110110	00101101	01100010	01101001	01100111	00100000	01101111	01101110	01100101	01101011	00101110
FONT CODE	X-POSITION	Y-POSITION	X-INCREMENT	b	i	g	SPACE	o	n	e	s	•

00000000	00000001	10101111	10110110	00101100	01001001	01110100	00100000	01101001 ...
NEW ENTITY	FONT CODE	X-POSITION	Y-POSITION	X-INCREMENT	I	t	SPACE	i

c

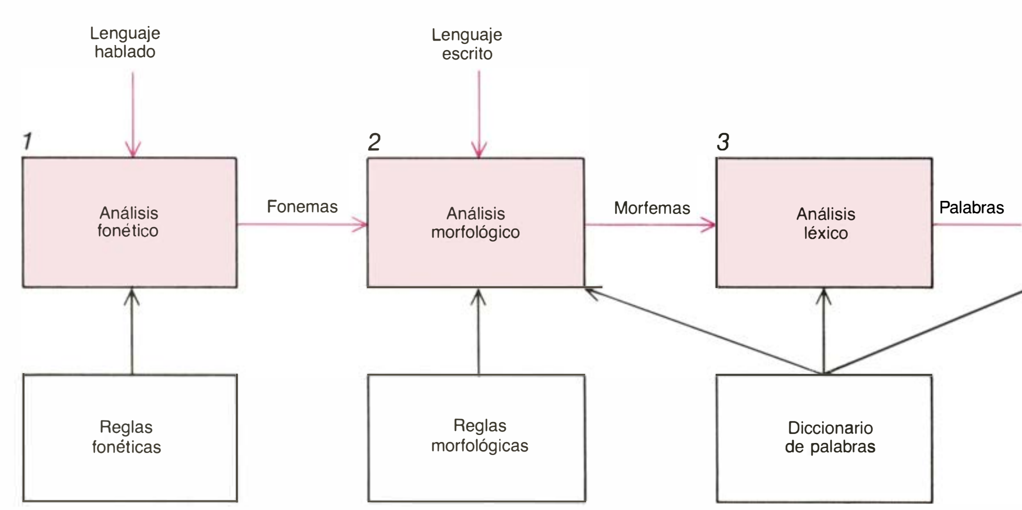
This is a sample of a *justified* piece of text, which contains small letters **and** big ones. It includes foreign words such as "peña"—which is Spanish—and foreign letters like α and $\aleph$, which can be baffling, and includes one wide space.

8. POR PROCESAMIENTO DE TEXTOS se entiende la preparación y redacción definitiva de un texto con la ayuda del ordenador. Para su realización, el proceso necesita varias representaciones del texto, porque el formato considerado ideal para las interacciones entre la programación y el usuario no sirve para enviar instrucciones a una máquina de tipografía, ni permite ofrecer una visión anticipada de lo que resultará tras el trabajo tipográfico. En el lenguaje TEX de composición tipográfica, la entrada tecleada por el usuario (*a*) incluye órdenes que especifican caracteres inhabituales, cambian el estilo de la letra de imprenta, marcan márgenes, etcétera. Tales órdenes se distinguen por una barra inclinada (\ ). La programación TEX "compila" la entrada, lo que produce un código máquina que excitará una máquina impresora (*b*). Con este propósito, el código se divide en distintas "entidades" cada una de las cuales especifica el tipo y la posición inicial de una secuencia de palabras. Los "incrementos *X*" codificados espacían las palabras hasta cubrir la distancia entre márgenes de la página impresa; dichos incrementos "justifican" la impresión por líneas. La página impresa (*c*) ilustra los resultados.

"posibilitando" e incluso de "imposibilitar" e "imposible". Para reconocer una raíz común en las palabras de este tipo se requiere un análisis morfológico, que puede acometerse con las técnicas desarrolladas para la traducción por ordenador. Propia de la computación es la invención de métodos para almacenar y buscar en un diccionario de sinónimos o en uno normal, que, para ser útiles, deberán alcanzar una extensión notable.

Las ayudas a la corrección cubren la ortografía, la gramática e incluso elementos del estilo. Los programas más sencillos de este tipo pretenden ajustar cada palabra de un texto con otra de un diccionario computarizado. Las palabras que no se corresponden con otras del diccionario quedan señaladas como posibles errores. Otros programas buscan los errores gramaticales comunes o los defectos del estilo. En este sentido, el "Writer's Workbench", desarrollado en los Laboratorios Bell del AT & T, incluye programas que buscan las palabras duplicadas como "la la" (un error frecuente de mecanografía), los errores de puntuación como "?." y los giros largos como "en este preciso momento". Otra ayuda a la corrección señala las "frases ampulosas" como "muestra una tendencia" y "llegar a una decisión" y sugiere sustitutos más sencillos, como "tiende" y "decidir". Y otra de estas ayudas busca los términos que especifican el género como, por ejemplo, "mailman" (cartero) y "chairman" (presidente) y sugiere que sean remplazados por "mail carrier" y "chairperson". (En español, el artículo sirve para deshacer cualquier ambigüedad de género.)

Además de cribar el texto en búsqueda de cadenas específicas de caracteres, hay programas de corrección que elaboran análisis estadísticos. Calculando la longitud media de las oraciones, la longitud media de las palabras y otros parámetros parecidos, computan un "índice de legibilidad". Los pasajes que tengan índices bajos le serán advertidos al escritor. Todavía no existe un programa que haga un análisis gramatical comprehensivo de un texto, si bien el Epistle, un sistema experimental de-



9. LA COMPRENSION AUTOMATIZADA DEL LENGUAJE requiere que el ordenador se sirva de varios tipos de datos almacenados (*recuadros blancos*) y que acometa distintos niveles de análisis (*recuadros en color*). Si el lenguaje es hablado, el primer análisis es fonológico (*1*): el ordenador analiza las ondas sonoras. Si el lenguaje es escrito, el primer análisis será morfológico (*2*): el ordenador descompone cada palabra en su radical, o forma básica, y las flexiones (por ejemplo: *ando*). Sigue después el análisis léxico

sarrollado en la Internacional Business Machines Corporation, toma algunas decisiones gramaticales. Emplea una gramática de 400 reglas y un diccionario de 130.000 entradas. Como todo soporte lógico que pretende analizar un texto sin ocuparse de su sentido, hay muchas frases que no pueden ser analizadas correctamente.

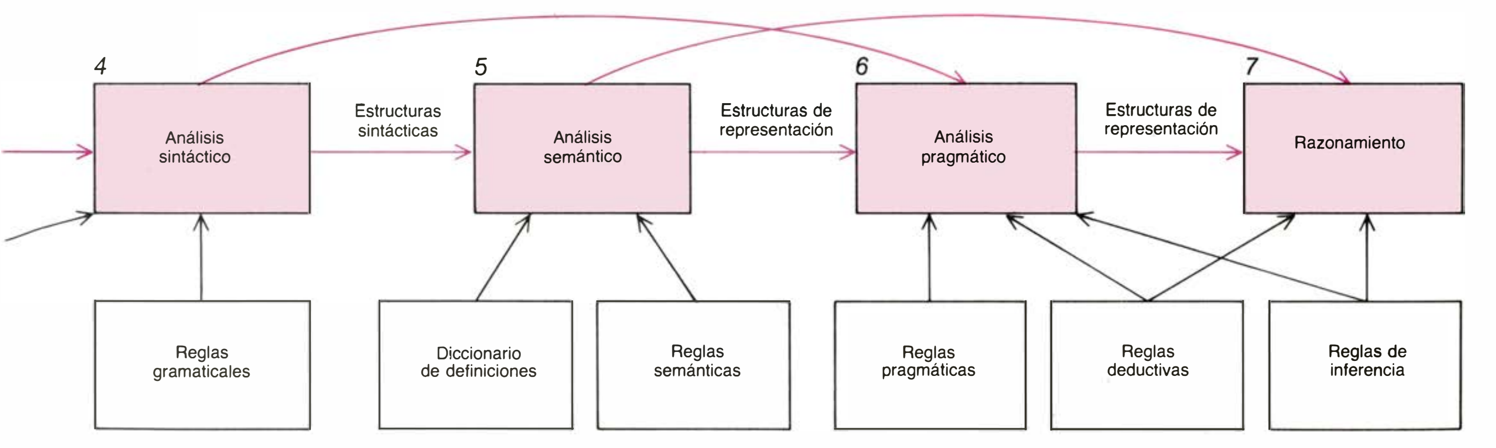
¿Hay algún programa que se ocupe realmente del significado, que muestre el mismo tipo de razonamiento que usaría una persona en el proceso lingüístico de la traducción, resumen o contestación a una pregunta? Semejante diseño ha constituido el sueño de la investigación en inteligencia artificial desde mediados de los años 60, cuando el equipo físico del ordenador y las técnicas de programación necesarios empezaron a aparecer, a la vez que la imposibilidad de la traducción por ordenador se hacía patente. Muchas son las aplicaciones que se beneficiarían de esa creación: programas que aceptan órdenes del lenguaje natural, programas que recogen y suministran información, programas que resumen textos y programas que adquieren conocimientos, basados en el lenguaje, para los sistemas expertos.

No existe soporte lógico que maneje el significado en un subconjunto importante del inglés. Todos los programas experimentales se fundan en buscar una versión simplificada del lenguaje y del significado y en probar qué se puede hacer dentro de sus confines. Algunos investigadores no perciben ninguna barrera fundamental contra el desarrollo de programas que alcancen una comprensión plena del lenguaje natural. Otros sostienen que la comprensión computarizada del lenguaje es imposible. Para seguir sus razonamientos, importa examinar las grandes líneas del funcionamiento de un programa que comprenda el lenguaje.

Un programa tal de comprensión del lenguaje necesita varios componentes, que se corresponden con los distintos niveles en los que se analiza el lenguaje [*véanse las figuras 9, 10, 11 y 12*]. La mayoría de los programas abordan el lenguaje escrito; se evita así el análisis de las ondas sonoras y el primer nivel a considerar es el morfológico. El programa utiliza unas reglas que descomponen una palabra en su radical, o forma básica, y sus flexiones, como las terminaciones "*s*" y "*ando*". Las reglas corresponden, sobre todo, a las reglas de ortografía que los niños aprenden en la escuela. Aprenden, por ejemplo, que el radical de "bailando" es "bail-" y el de "cocinar" es "cocin-". Una lista de excepciones se ocupa de las palabras que escapan de la norma, como las formas del verbo "ser". Otras reglas asocian las flexiones con "rasgos" del verbo. Por ejemplo "estoy bailando" es un verbo progresivo: indica que se está desarrollando una acción.

Para cada radical que resulta del análisis morfológico, el diccionario ofrece un conjunto de categorías léxicas a las que pertenece dicha raíz. Esto es el segundo nivel de análisis que efectúa el ordenador. Algunos lexemas (por ejemplo "the" en inglés y "nunca" en castellano sólo tienen una categoría léxica: otras tienen varias. "Dark" puede ser un nombre ("obscuridad") o un adjetivo ("obscuro"), como en castellano "naranja". "Bloom" puede ser un nombre ("flor") o un verbo ("florecer"); en español, cantar es verbo y nombre. En algunos casos el análisis morfológico limita las posibilidades. (En sus usos comunes "bloom" puede ser un nombre o un verbo, pero "blooming" sólo puede ser verbo en su modo gerundio.) El resultado del análisis morfológico y léxico es, por tanto, una secuencia de las palabras en la oración, llevando consigo cada voz una cantidad de información sobre sus rasgos y también información del diccionario. Este resultado sirve, a su vez, de asiento para un tercer componente del programa, el analizador sintáctico, que aplica las reglas de la gramática para determinar la estructura de la oración.

En el diseño de un buen analizador sintáctico se plantean dos problemas distintos. Primero: especificar un conjunto preciso de reglas –una gramática– que determine el conjunto de estructuras de frase posibles en un idioma. A lo largo de los últimos 30 años, el grueso de la investigación en lingüística teórica se ha centrado en el diseño de sistemas lingüísticos formales: construcciones en donde las reglas sintácticas de un lenguaje se expresan con precisión suficiente para que un ordenador pueda usarlas en el análisis del lenguaje. Las gramáticas generativo-transformacionales ideadas por Noam Chomsky, del Instituto de Tecnología de Massachusetts, constituyeron el primer intento totalizador. Especifican la sintaxis de un lenguaje mediante un conjunto de reglas cuya aplicación me-



(*3*), en el cual el ordenador pone las palabras en sus categorías léxicas (por ejemplo, nombre) e identifica determinados "rasgos", plurales por ejemplo. Viene a continuación el análisis sintáctico (*4*): la aplicación de las reglas gramaticales para sentar la estructura de la oración. Procede luego el análisis semántico (*5*). La frase se convierte en una forma que la haga susceptible de sacar inferencias. La última etapa es la del análisis pragmático (*6*): pone sobre el tapete el contexto de la frase; verbigracia, relación que existe entre el momento en que se habla y el momento a que se refiere lo expuesto. El ordenador puede ahora extraer inferencias (*7*), quizás a modo de preparación para responder a la frase proferida. (La ilustración es de Hank Iken.)

cánica engendra todas las estructuras permitidas.

Segundo problema: el propio análisis sintáctico. Al topar con una parte de una frase, no siempre es posible decir exactamente qué papel desempeña en la oración, ni si las palabras que la componen deben ir juntas. Tomemos la frase "Roses will be blooming in the dark gardens we abandoned long ago" ("*Florecerán las rosas en los jardines obscuros que abandonamos hace tiempo*"). Las palabras "in the dark" pueden tomarse como una frase completa: después de todo están bien combinadas (gramaticalmente) y tienen sentido ("en la obscuridad"). Pero la frase no puede formar una unidad coherente en el análisis completo de la frase entera, porque entonces nos obliga a interpretar "Roses will be blooming in the dark" como una oración completa, dejando "gardens we abandoned long ago" sin función en la oración.

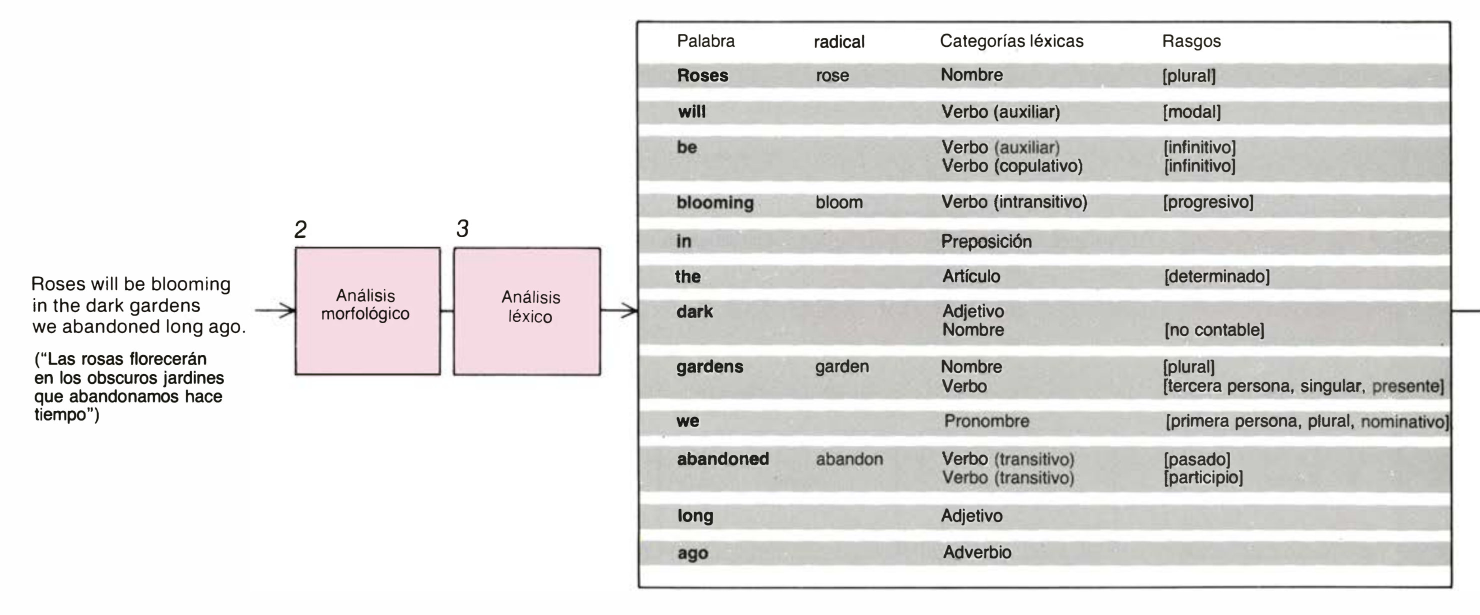
Los analizadores sintácticos adoptan varias estrategias para explorar las múltiples maneras en que se pueden componer las frases. Algunos trabajan de arriba abajo, tratando de encontrar frases posibles desde el principio; otros proceden de abajo arriba, tanteando con combinaciones locales de palabras. Los hay que desandan el camino y exploran en profundidad las alternativas si resulta que les falla una vía; y también están los que siguen procesos paralelos rastreando simultáneamente distintas posibilidades. Algunos usan formalismos desarrollados por los lingüistas (como la gramática transformacional). Otros emplean formalismos más recientes que fueron diseñados para su aplicación en los ordenadores. Estos últimos son más adecuados para el análisis sintáctico. En este sentido, las "redes de transiciones aumentadas" expresan la estructura de las frases y de las oraciones subordinadas como una secuencia explícita de "transiciones" a seguir por el ordenador. Las "gramáticas de función léxica" crean una "estructura funcional" donde las funciones gramaticales como sujeto y complemento quedan ligadas explícitamente a las palabras o frases que desempeñan esas funciones.

Aunque ninguna gramática formal soluciona con éxito los problemas gramaticales de ningún lenguaje natural, el soporte lógico existente (gramáticas y analizadores sintácticos) puede manejar más del 90 por ciento de todas las oraciones. Pero no se trata de un rendimiento neto. Las frases admiten cientos y hasta miles de análisis sintácticos posibles. La mayoría de ellos carecerán de sentido coherente. La gente no es consciente de la consideración y rechazo de tales análisis incoherentes, pero los programas analizadores se ven inundados por alternativas que carecen de sentido.

El fruto o salida de un programa de análisis se convierte en la entrada para el cuarto componente de un programa de comprensión del lenguaje: el analizador semántico; éste traduce la forma sintáctica de una oración en su forma "lógica". Se pretende colocar las expresiones lingüísticas en una forma que permita al ordenador aplicar procedimientos de razonamiento y sacar conclusiones. De nuevo existen aquí teorías opuestas sobre cuál sea la representación más apropiada. Como en el caso del análisis sintáctico, lo principal es la eficacia y el rendimiento.

La eficacia depende del hallazgo de las estructuras formales apropiadas para codificar el significado de las expresiones lingüísticas. Una posibilidad es el cálculo de predicados, que emplea el cuantificador universal $\forall$ para indicar "todo", y el cuantificador existencial $\exists$, para indicar "hay". En el cálculo de predicados "las rosas florecerán" equivale a la afirmación "existe algo que es una rosa y que ésta florece...". Esto acarrea un problema. ¿Es una rosa suficiente para representar lo significado por "las rosas florecerán"? ¿Cómo puede decidir el ordenador? El dilema se enreda si la frase incluye un nombre de cosas no contables o numerables, como "el agua" en "el agua fluirá". No se puede particularizar en absoluto el agua. Al diseñar una estructura formal para el significado de expresiones lingüísticas surgen muchos problemas parecidos, que se derivan de la vaguedad inherente del lenguaje.

Hay que prestar también atención al rendimiento. El ordenador empleará la forma lógica de una oración para sacar inferencias que ayuden, a su vez, al análisis del significado de la frase y a la formulación de la respuesta. El cálculo de predicados y otros formalismos no son directamente amoldables a una



Palabra	radical	Categorías léxicas	Rasgos
Roses	rose	Nombre	[plural]
will		Verbo (auxiliar)	[modal]
be		Verbo (auxiliar) Verbo (copulativo)	[infinitivo] [infinitivo]
blooming	bloom	Verbo (intransitivo)	[progresivo]
in		Preposición	
the		Artículo	[determinado]
dark		Adjetivo Nombre	 [no contable]
gardens	garden	Nombre Verbo	[plural] [tercera persona, singular, presente]
we		Pronombre	[primera persona, plural, nominativo]
abandoned	abandon	Verbo (transitivo) Verbo (transitivo)	[pasado] [participio]
long		Adjetivo	
ago		Adverbio	

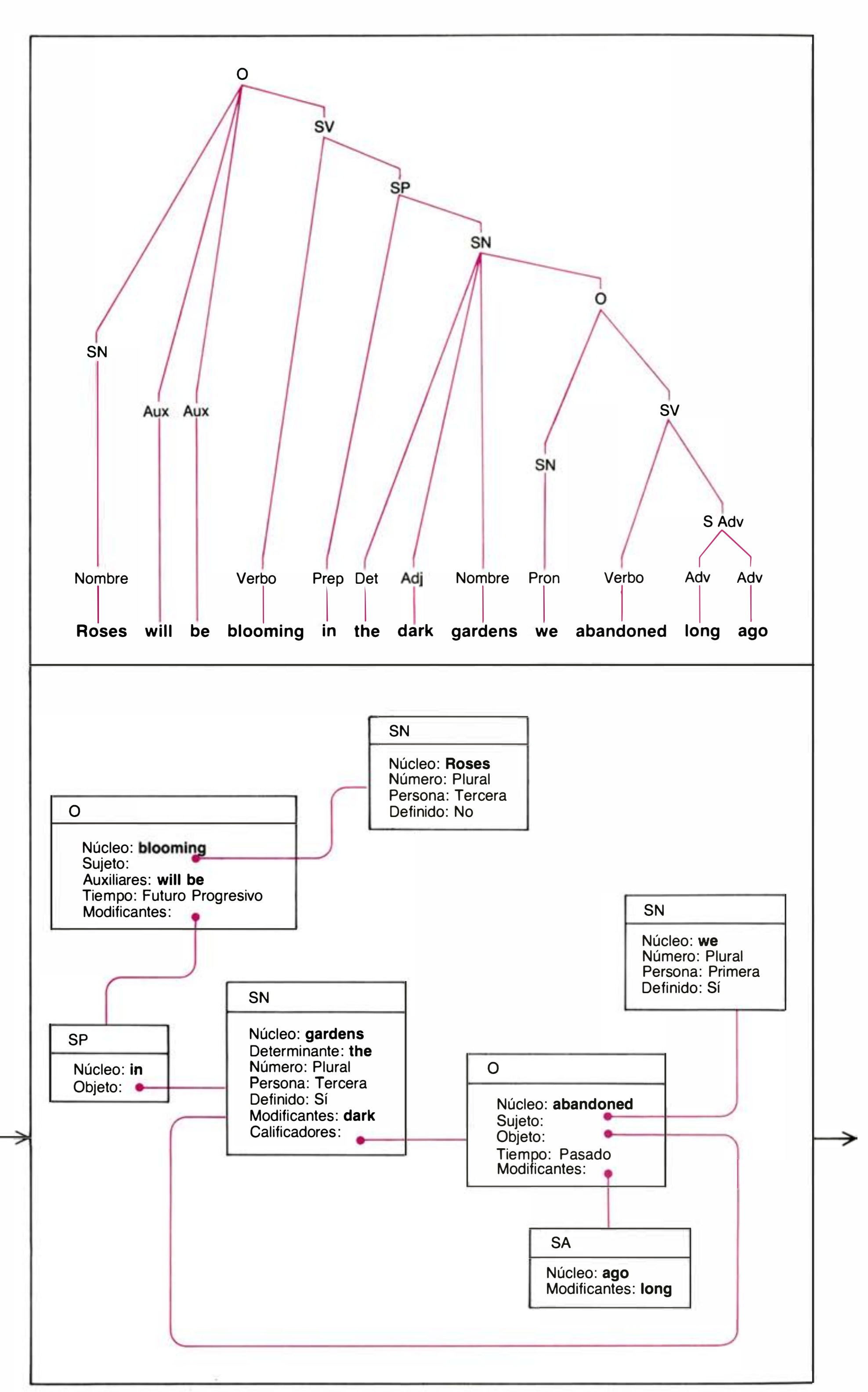
10. SUCESION DE ANALISIS desarrollados por un programa de ordenador hipotético. Nos sugiere cómo funciona la programación que "entiende" el lenguaje. En esta ilustración, se ha dado al programa la oración "Roses will be blooming in the dark gardens we abandoned long ago" ("Las rosas florecerán en los obscuros jardines que abandonamos hace tiempo"); los primeros análisis (morfológico y léxico) aportan la lista de las palabras con sus radicales, sus categorías léxicas y sus rasgos. "Blooming", por ejemplo, es un verbo progresivo: significa un acto en progreso. Los datos sirven de entrada para el nivel sintáctico del análisis: el análisis sintáctico de la oración. Aquí pone la estructura superficial o gramatical de "Roses will be blooming..." en la forma de un

computación eficaz. Pero han aparecido representaciones más "procedimentales". Imaginemos que se tenga que responder a la pregunta "¿Hay flores en los jardines que abandonamos hace mucho tiempo?" El ordenador necesita saber que las rosas son flores. Este conocimiento podría ser representado por una fórmula del cálculo de predicados que equivaliera a la afirmación "todo lo que es una rosa es una flor". El ordenador podría emplear entonces técnicas desarrolladas para la demostración automática y operar así la deducción necesaria. Otro enfoque sería conceder a determinadas inferencias un estatuto de proceso privilegiado. Por ejemplo, las deducciones clasificatorias básicas podrían representarse directamente en estructuras de datos [*véase la figura 12*]. Se necesitan continuamente tales deducciones para razonar sobre las propiedades normales de los objetos. Otros tipos factuales podrían representarse entonces de una forma más cercana al cálculo de predicados (por ejemplo, que las flores necesitan agua para crecer). El ordenador podría servirse de ambos para sacar inferencias (verbigracia, si las rosas no tienen agua, no crecerán).

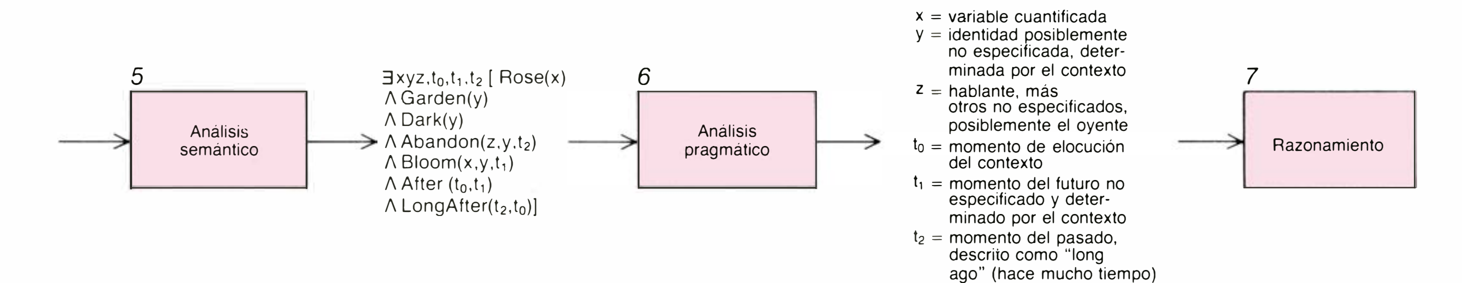
Se ha investigado mucho en lo relativo al diseño de los "lenguajes de representación" que posibilitan la codificación eficaz y eficiente del significado. La mayor dificultad estriba en la naturaleza del razonamiento humano de sentido común. La mayor parte de lo que sabe una persona no puede formularse en reglas lógicas de todo o nada; este saber descansa en las "expectativas normales". Si se pregunta "¿Hay polvo en el jardín?" la respuesta será afirmativa, casi con toda seguridad. Pero este "sí" no será nunca una inferencia lógica; algunos jardines son hidropónicos, donde las plantas crecen en el agua. La gente se guía por las expectativas normales, sin pensar en las excepciones a menos que sean relevantes. Pero apenas si se ha avanzado en el camino de la formalización del concepto de "relevancia" y del modo en que afecta el trasfondo de las expectativas aplicadas

4

Análisis sintáctico



árbol. El ordenador rechaza probablemente varios árboles incorrectos. Por ejemplo, desechará el árbol que considere una frase entera "Roses will be blooming in the dark". Pone la estructura profunda de "Roses will be blooming" en un diagrama de estructura funcional. Así se hacen explícitas las relaciones entre las partes de una oración; se ilustran por líneas que unen recuadros. Algunas relaciones era explícitas en la estructura superficial (por ejemplo que "roses" es el sujeto de "blooming"). Otras no lo eran (por ejemplo que "gardens" es el objeto de "abandoned"). El análisis sintáctico llega en las últimas etapas del programa. El desarrollo del análisis sintáctico debe mucho a las gramáticas generativo-transformacionales ideadas por Chomsky.



11. LOS ANALISIS CONCLUYEN con la conversión de la estructura sintáctica de "Roses will be blooming..." en una forma a partir de la cual el ordenador puede sacar inferencias. En este ejemplo, la conversión se funda en el cálculo de predicados; así el módulo de análisis semántico de la programación hipotética representa el contenido lógico de "Roses will be blooming..." por símbolos que pueden ser traducidos como "*x* es una rosa e *y* es un jardín e *y* está oscuro". Finalmente, el módulo de análisis pragmático especifica lo que se sabe acerca de las variables *x*, *y*, *z*, t_0, t_1 y t_2. La variable *x*, por ejemplo, está "cuantificada": afirma la existencia de algo, sin identificarlo en particular. En otros términos, el ordenador supone que "roses" se refiere a las rosas en general, no a algunos ejemplares particulares. De aquí que las rosas no es un nombre "definido". (Se tomó esa decisión en el curso del análisis semántico.) Por otra parte, *z* mantiene su ambigüedad porque representa el pronombre ambiguo "we", *nosotros*. (Representación esquemática de Hank Iken.)

en la comprensión de las expresiones lingüísticas.

La última etapa del análisis de un programa de comprensión del lenguaje se refiere al análisis pragmático: el análisis del contexto. Todas las oraciones están integradas en un marco: proceden de un hablante específico en un momento específico y remiten, al menos implícitamente, a un cuerpo de conocimientos. Algunas cosas del marco son sencillas: el pronombre "yo" señala el hablante; el adverbio "ahora" designa el momento cuando se profiere la frase. Mas incluso éstas pueden resultar problemáticas. Piénsese en el uso de la palabra "ahora", que escribo hoy en una carta que usted leerá dentro de tres o cuatro días. De cualquier manera, programas bastante sencillos pueden sacar las conclusiones correctas la mayoría de las veces. Otras situaciones son más difíciles. El pronombre "nosotros" es un ejemplo; "nosotros" puede referirse al hablante y al receptor o al hablante y a otra tercera persona. Ninguna de las dos posibilidades (ni quién puede ser la tercera persona) queda explícitamente reflejada; de hecho es una fuente habitual de malentendidos en la conversación.

Otros tipos de implicaciones no están señalados por palabras tan traidoras como "nosotros". La frase "Las rosas florecerán..." presupone la identificación de algún momento en el futuro cuando las rosas realmente echen flor. La frase pudo haber seguido a estas otras: "¿Cómo estará cuando lleguemos a casa?" o "Se acerca el verano". De un modo parecido, el sintagma nominal "los jardines oscuros que abandonamos hace mucho tiempo" tiene un significado dependiente del contexto. Puede que existan sólo unos jardines donde hayamos estado juntos; o puede haber varios. La frase presupone un cuerpo de conocimiento a partir del cual identificamos los jardines de que se está hablando. Lo reseñable es que una frase que empieza con un artículo determinado especifica pocas veces el objeto al cual se refiere.

Expresiones de ese tipo se abordan codificando los conocimientos del mundo en una forma que el programa pueda utilizar para sacar inferencias. Por ejemplo, de la oración "Fui a un restaurante y el camarero estuvo descortés" puede inferirse que "el camarero" se refiere a la persona que sirvió la comida del hablante si sus conocimientos incluyen un guión, por decirlo así, de los sucesos típicos que se desarrollan durante una comida en un restaurante. (El cliente suele ser servido por un determinado camarero o camarera.) En los casos más complejos, viene en nuestra ayuda el análisis de los propósitos y las estrategias del hablante. Si se oye "Mañana me examino de matemáticas, ¿dónde está el libro?" se puede suponer que el hablante piensa estudiar y que con "el libro" alude al texto de matemáticas empleado en un curso que sigue el propio hablante. Este enfoque topa con la misma dificultad que entorpece la representación del significado: la dificultad de formalizar el trasfondo de sentido común que determina qué propósitos y qué estrategias son pertinentes y cómo se influyen mutuamente. Los programas escritos hasta ahora funcionan sólo en campos muy artificiales y limitados; y no está claro hasta dónde pueden extenderse tales programas.

Más problemáticos son los efectos del contexto sobre el sentido de las palabras. Supóngase que al intentar entender "los jardines obscuros que abandonamos hace mucho tiempo", nos disponemos a aplicar un sentido específico a la palabra "obscuro". ¿Cuál debería ser? ¿El "obscuro" de "esos días obscuros de tribulación", el de "¡qué obscuro está cuando se apagan las luces!" o el de "colores obscuros"? Aunque un núcleo de similaridad une los distintos usos de un término, su significado pleno viene determinado por el uso concreto y por la comprensión previa que espera el hablante del oyente. "Los jardines obscuros" podría tener un sentido definido para la persona a quien se dirige el mensaje; para los demás queda bastante "obscuro".

Pudiera parecer viable, a primera vista, distinguir entre los usos "literales" del lenguaje y los metafóricos o poéticos. Los programas de ordenador que afrontan exclusivamente el lenguaje literal podrían ser liberados de los dilemas de contexto. Pero el problema estriba en que las metáforas y "el significado poético" no se ciñen a los textos literarios. El lenguaje ordinario está saturado de metáforas inconscientes, como cuando se dice "He perdido dos horas intentando hacerme entender". Casi todas las palabras tienen un campo abierto de significados, extendiéndose poco a poco desde las que parecen ser literales del todo hasta las indudablemente metafóricas.

Las limitaciones que encuentra la formalización del significado del contexto hacen imposible ahora –y quizá por siempre– diseñar programas de ordenador que remeden la comprensión humana del lenguaje. Los únicos programas en servicio que intentan hoy una comprensión, siquiera limitada, son los limítrofes ("front-ends") de lenguaje natural, que facultan al usuario de un programa para solicitar información preguntando en inglés. El programa responde en inglés o con una presentación de datos en la pantalla.

Ejemplo de un primer programa de éstos es SHRDLU, desarrollado a finales de los años sesenta. Recurriendo al mismo, una persona entabla comunicación con el ordenador en inglés en torno a un mundo simulado de bloques colocados encima de una mesa. El programa analiza las preguntas, las órdenes y las instrucciones que emite el usuario, y responde con las palabras apropiadas o con acciones desarrolladas en la escena simulada. El éxito de SHRDLU debióse, en parte, a su tema de conversación, circunscrito a un dominio sencillo y especializado: los bloques y unas cuantas acciones que pueden realizarse con ellos.

Recientemente se han diseñado interfaces de "programas limítrofes" guiados por una finalidad práctica. La persona que desea acceder a la información almacenada en el ordenador escribe frases en lenguaje natural que el ordenador interpreta como preguntas. El alcance de las preguntas está limitado por la gama de datos a partir de los cuales se emiten las respuestas; así se puede dar un significado preciso a las palabras. En una base de datos sobre automóviles, por ejemplo, "obscuro" puede definirse como color "negro" y "azul marino", y nada más. Posee el significado de contexto, pero está predeterminado por el constructor del sistema, y el usuario debe aprenderlo.

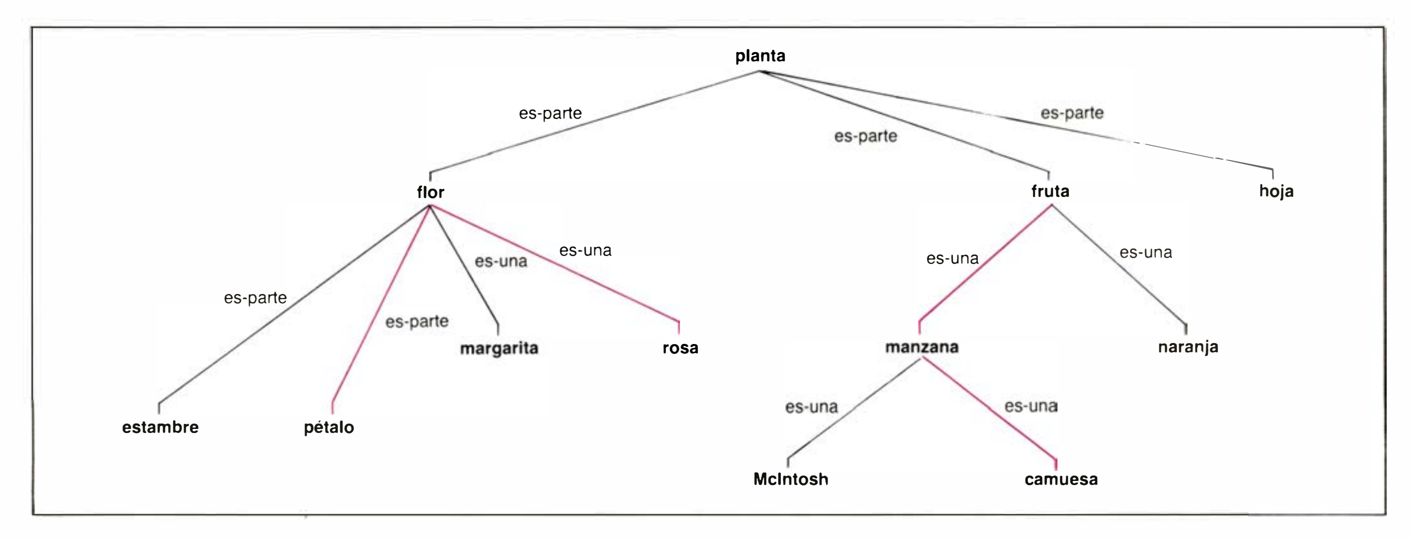
La ventaja principal que encierran los "programas limítrofes" de lenguaje natural es la suave barrera inicial que presentan a los usuarios. La persona invitada a hacer una pregunta en inglés suele aceptar, y si el ordenador se nuestra incapaz de comprender la forma específica de la pregunta, el usuario no tendrá inconveniente en alterar la frase hasta que la interacción funcione. Con el tiempo, el usuario dominará las restricciones impuestas por el sistema. Por contra, la persona obligada a aprender un lenguaje especializado para formular una pregunta pensará que se le exige un trabajo excesivo.

Consideremos, por último, otro tipo de sistema, bastante nuevo, denominado coordinador. Sustituye el correo electrónico tradicional por un proceso que ayuda a la generación de mensajes y que controla el progreso de las conversaciones resultantes. Los coordinadores se fundan en la teoría de la enunciación o acto del habla, que afirma que todas las expresiones se distribuyen en un pequeño número de categorías; algunas enunciaciones son meras constataciones: "Está lloviendo". Otras son expresivas: "Siento haberle pisado". Otras, ruegos: "Por favor, ayúdale con el paquete" o "¿Cómo se llama usted?". Otras son compromisos: "Lo haré mañana". Otras, declarativas: "Despedido". (Las declarativas se diferencian de las constataciones en el hecho de que entran en vigor en virtud de las propias palabras que las dictan.)

La clasificación de las enunciaciones es útil porque los actos del habla en las diversas categorías no ocurren al azar. Cada uno tiene "condiciones de felicidad", bajo las cuales es apropiado proferirlo, y "condiciones de satisfacción", bajo las cuales se cumple. Por ejemplo, un ruego o un compromiso llevan consigo, explícita o implícitamente, un plazo de tiempo dentro del cual deberían cumplirse. Además, cada enunciación es parte de una conversación que sigue un modelo regular. La regularidad es crucial para una acertada comunicación.

Como con todos los aspectos del lenguaje, la comprensión completa de cualquier enunciación está siempre inmersa en el trasfondo de esperanzas inarticuladas del hablante o del oyente. La expresión oral "Estaré aquí mañana" podría ser una predicción o una promesa y "¿Juega usted al tenis?" podría ser una pregunta o una invitación. En la conversación hablada, la entonación y el énfasis desempeñan un papel prominente en el establecimiento de tal significado.

Los sistemas coordinadores se ocupan de las enunciaciones incorporadas en mensajes especificando qué se necesita hacer y cuándo. El sistema no pretende, de suyo, analizar el contenido lingüístico de los mensajes. Por el contrario, el soporte lógico de procesamiento de palabras de la parte emisora pide al emisor que haga explícito el contenido de la enunciación de cada mensaje. Una persona podría escribir en el propio mensaje "Tendría mucho gusto en procurarle ese informe", pero debería añadir (tecleando un código especial) que el mensaje es un ACCEPT ("ACEPTACIÓN") de una determinada REQUEST ("PREGUNTA"). El sistema del ordenador puede seguir, entonces, la pista de los mensajes y sus interconexiones. En particular, el sistema puede controlar el desarrollo completo de las conversaciones, llamando la atención del usuario hacia los casos en los cuales está pendiente algo inmediato, o en los que no se ha cumplido una fecha convenida para la satisfacción.



12. LA RED SEMANTICA es una forma especializada de datos almacenados que representa las relaciones lógicas; el ordenador puede así sacar inferencias. Aquí, simplemente por seguir las pistas de las conexiones de la red (*color*) se ha deducido que una manzana camuesa es una fruta y que una rosa tiene pétalos. Hechos no representados por una red pueden describirse con otros mecanismos, por ejemplo, a través del cálculo de predicados.

Desde una perspectiva más amplia, los coordinadores son un miembro más de la gran familia de logicales que ofrece a los usuarios un medio estructurado en el que se amplía el lenguaje por indicaciones explícitas de cómo se vertebran las cosas. Otro tipo de soporte lógico de esta familia aporta los medios para subrayar y relacionar entre sí distintos documentos. Un tercer tipo consiste en tablones de anuncios computarizados merced a los cuales los usuarios almacenan y reciben mensajes no ordenados a un receptor específico. Los mensajes están "anunciados" con una estructura adicional que indica su contenido y facilita su localización por parte del lector interesado.

El pronóstico más obvio sobre el futuro de la programación relativa al lenguaje nos habla de la disminución del costo del soporte físico, con la consiguiente generalización venidera de aplicaciones que hoy son posibles, aunque inviables. Está por ver la programación que remede la plena comprensión humana del lenguaje. Algunas tendencias apuntan, sin embargo.

La primera es que el lenguaje hablado recibirá mayor atención. Es cierto que la comprensión computarizada del lenguaje hablado plantea todas las dificultades del lenguaje escrito y muchas más. La mera separación de una locución en sus palabras componentes puede "constituir un tormento" para el computador; vale decir, la esperanza de una máquina que escriba textos al dictado es al menos tan lejana como la esperanza de llegar a la traducción automática de alta calidad y a la comprensión automática del lenguaje. Muchos dispositivos útiles no necesitan, sin embargo, el análisis de habla encadenada. Los sistemas existentes que idenfican palabras o frases habladas de un vocabulario fijo mejorarán la comunicación entre los usuarios y las máquinas. La aparición reciente de pastillas de circuitos integrados, baratas, que procesan señales acústicas facilitará esta tendencia. Los sintetizadores de voz que generan locuciones comprensibles (aunque no den voz de sonido natural) desempeñarán, asimismo, un papel creciente. La mejor "comprensión" del habla y las técnicas de codificación convertirán en algo normal la anotación y mensajes acústicos.

Una segunda tendencia que se observa en la programación relativa al lenguaje se refiere a las restricciones sobre el dominio lingüístico, que se habrán de tratar teóricamente y manejar con mucho cuidado. Varias veces a lo largo del artículo he dado ejemplos de cómo los ordenadores abordan el significado de una manera admisible, porque operan en un dominio limitado de sentidos posibles. La gente que trabaja con este soporte lógico advierte en seguida que, si bien el ordenador no abarca el lenguaje en todo su alcance, el subconjunto disponible constituye una buena base para entrar en comunicación. El éxito comercial de la programación del futuro dependerá del descubrimiento de dominios donde, no obstante las restricciones sobre los posibles sentidos de una frase, goce el usuario de una amplia libertad en su forma de expresión.

Tercera tendencia: el desarrollo de sistemas que combinen lo natural y lo formal. Suele darse por sentado que el lenguaje es el mejor medio que la gente tiene para comunicarse con los ordenadores. Los planes para una "quinta generación" de ordenadores inteligentes se basan en esta premisa. Pero no es evidente en absoluto la validez del supuesto. En algunos casos, ni siquiera la inteligencia cabal del lenguaje natural alcanza la expresividad de un dibujo; en otros muchos, una comprensión parcial del lenguaje natural resulta menos aprovechable que una interfaz formal bien diseñada. Considérese el trabajo con "programas limítrofes" ("front-ends") de lenguaje natural. Aquí el lenguaje natural promueve la aprobación inicial del sistema, pero una vez que los usuarios se acercan hacia formas estilizadas de lenguaje descubren que pueden emplearlas sin preocuparse por si la máquina interpreta bien o no las instrucciones.

La mayoría de los sistemas que gozan hoy de éxito facilitan esta transición. Algunos (coordinadores incluidos) mezclan lo natural y lo formal: enseñan al usuario a reconocer las propiedades formales de las expresiones y a introducirlas explícitamente en los mensajes. El ordenador se ocupa de las estructuras formales, mientras que la gente manipula tareas donde importa el contexto y no pueden aplicarse reglas precisas. Otros sistemas incorporan un sistema muy estructurado de preguntas; así, el usuario, conforme va acumulando experiencia, comprueba que las formas artificiales le ahorran tiempo y problemas. No se le asignan, pues, al ordenador las tareas difíciles e indefinidas del análisis lingüístico; sirve, en cambio, como un medio lingüístico estructurado. Ese puede ser el medio más útil en que el ordenador trabaje con lenguajes naturales.

Programación de representaciones gráficas

Las imágenes interactivas, hasta ayer dominio de especialistas, constituyen cada vez más el medio de comunicación entre usuario y ordenador

Andries van Dam

Ivan E. Sutherland, pionero en la programación de ordenadores para la creación y manipulación de imágenes, comentaba a propósito de su actividad favorita: "Imagino las imágenes del ordenador como una ventana abierta al mundo de Alicia, al país de las maravillas; el programador tanto puede representar objetos que obedezcan las leyes de la naturaleza como otros que sigan leyes definidas en el programa. Las imágenes de ordenador me han permitido aterrizar sobre la cubierta de portaviones, observar el choque de partículas nucleares contra un pozo de potencial, pilotar un cohete casi a la velocidad de la luz e introducirme en el funcionamiento de los más recónditos circuitos de un ordenador"

Hasta ahora, sólo unos cuantos afortunados podían compartir las recientes experiencias de Sutherland con los mágicos poderes de los gráficos interactivos. En su mayoría eran científicos e ingenieros dedicados al diseño por ordenador, al análisis de datos y a los modelos matemáticos. El privilegio de la exploración de los mundos reales e imaginarios a través de la pantalla se está generalizando. De hecho, las representaciones gráficas constituyen, cada vez más, la manera convencional de comunicarse con los ordenadores.

Son diversas las razones de ese cambio. Primero, las espectaculares mejoras de la relación precio-prestaciones de ciertos componentes microelectrónicos, que permiten el diseño de terminales gráficos muy complejos, y la fácil adquisición de ordenadores personales orientados a la generación de imágenes. En particular, los avances en el diseño y fabricación de circuitos microelectrónicos han culminado en una nueva generación de "pastillas" con enorme capacidad de almacenamiento de información a un precio por unidad baratísimo. Gracias a ese desarrollo la técnica de generación de imágenes por "barrido" ha ganado en competitividad económica. Análoga al sistema que emplean los televisores, esa técnica rastrea, o barre, la pantalla en líneas horizontales. En los gráficos de barrido, cada "pixel", o elemento de imagen, se representa en la memoria del ordenador; ello permite controlarlos independientemente en los programas, lo que redunda en una máxima flexibilidad en la creación de imágenes.

Al propio tiempo, avances parejos en programación han extendido el campo de aplicaciones tratables por métodos gráficos. Los nuevos paquetes de programas para aplicaciones comerciales permiten, por ejemplo, la visualización de datos en forma de gráficos o diagramas hasta en los más sencillos ordenadores personales. En otro orden, los paquetes de programas para gráficos de alto nivel se están abaratando, a la vez que ganan en sencillez la escritura de programas y su adaptación de un ordenador a otro.

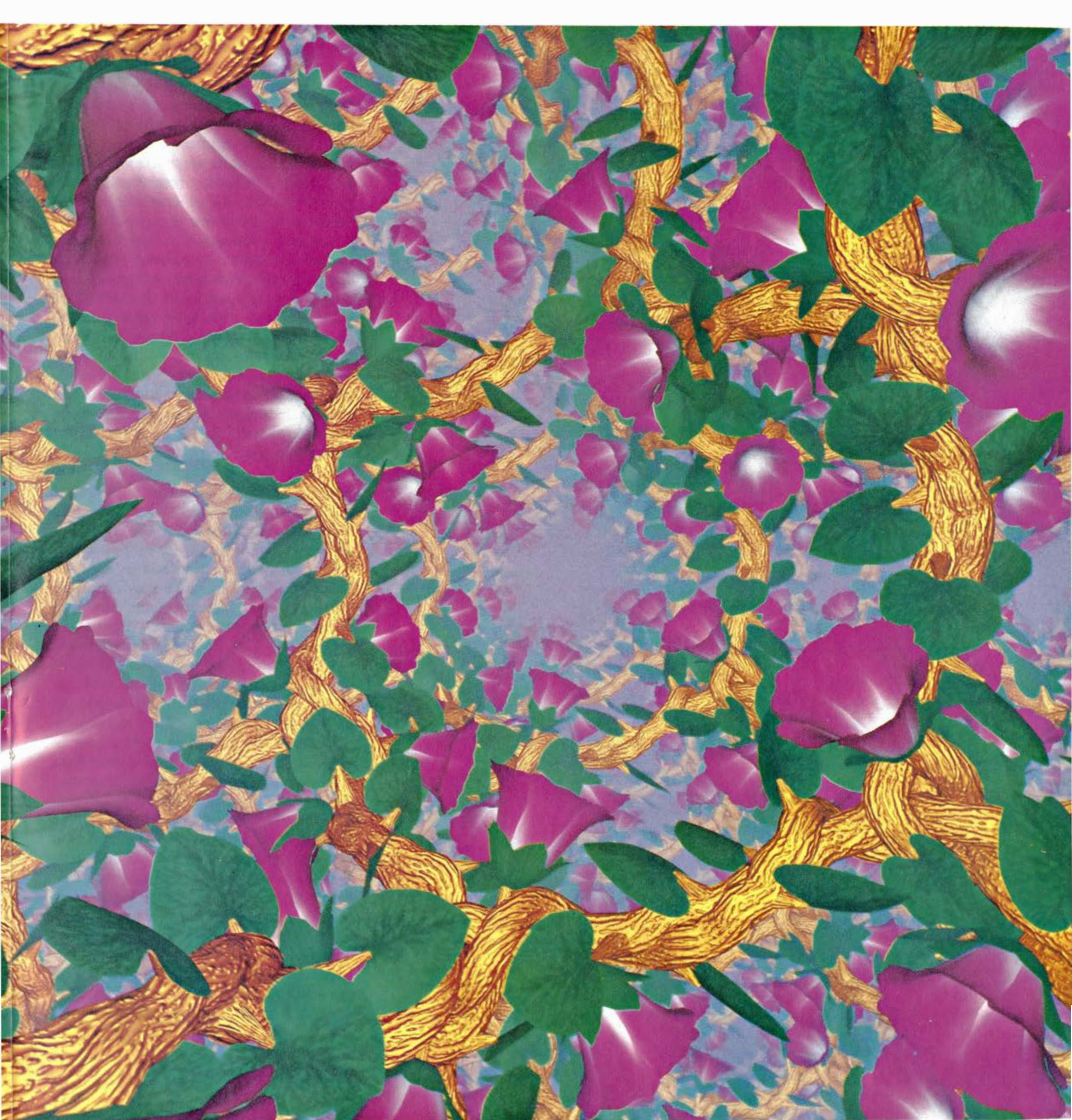
En la creciente popularización de las representaciones gráficas por ordenador ha intervenido también la contribución de las imágenes a la creación de interfases operador-máquina de manejo cómodo, lo que se ha dado en llamar interfases "amistosas con el usuario" (*user-friendly*). La expresión remite a una nueva filosofía de la programación, especialmente bien ilustrada por el conjunto de técnicas desarrolladas en la década de 1970 en el Centro de Investigación de la Corporación Xerox de Palo Alto. Las imágenes que se apoyan en ese enfoque (influido por los trabajos pioneros de Douglas C. Engelbart, del Instituto de Investigación de Stanford) se emplean ya en paquetes comercializados, desde el ordenador Star, de la Xerox, hasta el ordenador personal Mcintosh, de Apple Computer Inc. Ese tipo de interfases se distingue por la metáfora del "escritorio"; esto es: la pantalla se divide en varias zonas, llamadas ventanas, que pueden solaparse. La imagen trata de simular un escritorio cubierto de papeles. Cada ventana puede visualizar la aplicación de un programa; así el usuario trabaja simultáneamente con material gráfico y textos. Además, con ayuda de la si-

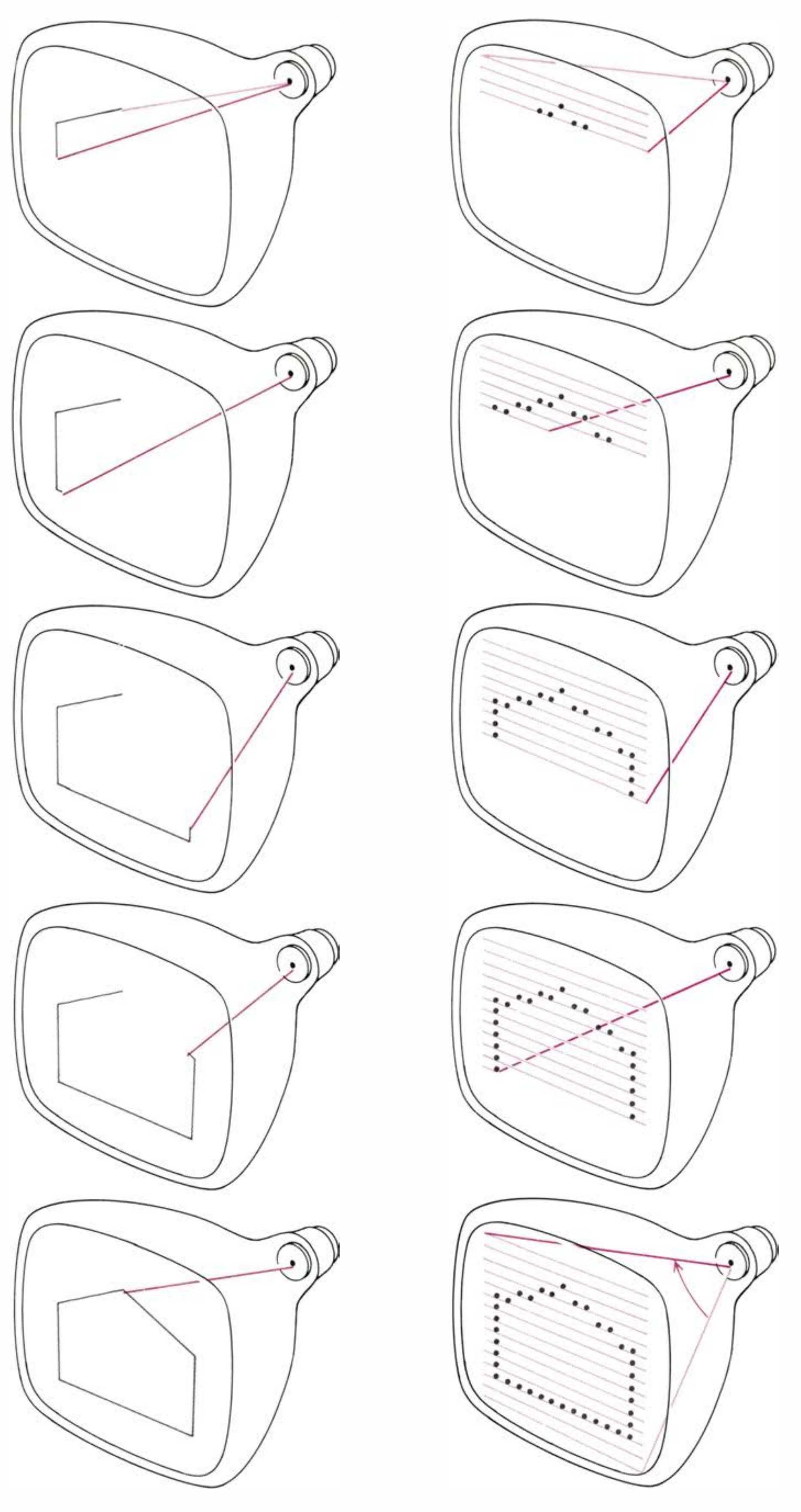
1. MUNDO IMAGINARIO DE PARRAS Y FLORES creado en el ordenador por Ned Greene, del Instituto de Tecnología de Nueva York. Las parras siguen los bordes de una red tridimensional análoga a la estructura cristalina del diamante. La imagen es un fotograma de una secuencia animada en la que el punto de observación desciende por uno de los pasillos que dejan entre sí las parras. Los objetos de la escena se definieron primero como redes de polígonos. Las parras se plasmaron por medio de una técnica que produce la impresión de relieve, ajustando el grado de matización según la información de profundidad extraída de imágenes de rayos X tomadas de un molde en yeso de una verdadera corteza de árbol. Se dio color a las hojas y los sépalos de las flores disponiendo antes sobre su representación en redes ciertos valores de color previamente almacenados, todo ello según la técnica denominada "cartografiado de texturas". Para producir la suave degradación cromática de los pétalos se asignó el color a los vértices que correspondían a los pétalos en la representación en redes; el programa de plasmación de la imagen interpoló los colores de los vértices a lo largo de las facetas poligonales. La sensación de bruma se logró reduciendo el contraste según una función exponencial de la distancia al observador. "Sin bruma, o alguna otra señal de profundidad, comenta Greene, la escena resulta prácticamente incomprensible". La imagen está compuesta por unos 1,9 millones de polígonos, cuya plasmación requirió 18 horas de trabajo a un miniordenador vax 11/780 de la Digital Equipment Corporation. Cada pixel, es decir, elemento de imagen, del patrón de barrido de 1536 por 1536 requiere 24 bits de información. Jules Bloomenthal, Paul Heckbert y Lance Williams han escrito también programas para el proyecto.

mulación de operaciones como "cortar y pegar" pueden tejerse los distintos elementos en un solo documento.

Los nuevos sistemas de manejo "amistoso" suelen apoyarse en el enfoque WYSIWYG (*what you see is what you get*, lo que ves es lo que va a salir), según el cual la representación que aparece en pantalla semeja en lo posible lo que se imprimirá o se copiará en cualquier otro soporte duro. Para diseñar una página de un texto, por ejemplo, ya no hay que picar códigos de formato (como *.pp* para indicar "párrafo", o *.s2*, para "saltar dos líneas") que interpretará un programa de generación de formatos cuando el usuario concluya la revisión del texto. Por el contrario, márgenes y sangrías se ajustan trabajando con un facsímil que lleva señalados los espacios. El texto se ajusta continuamente al formato a medida que se revisa. Puesto que los caracteres se generan por medio de programas de tratamiento de gráficos, pueden visualizarse a casi cualquier cuerpo o tipo, espaciados todos por un igual o de acuerdo con su anchura. Se pueden manejar igualmente símbolos matemáticos, alfabetos no latinos y hasta ideogramas chinos y japoneses.

Los programas de aplicación de esos sistemas se basan en un conjunto uniforme de instrucciones epecíficas. Por ejemplo, en vez de escribir una serie de comandos en un teclado alfanumérico, pueden elegirse los que interesen de varios menús, o listas, mostrados en pantalla. Una instrucción se ejecutará sin





2. DOS SON LOS METODOS empleados para producir una imagen en una pantalla de rayos catódicos. En el vectorial (*izquierda*) el haz de electrones se dirige continuamente entre dos puntos cualesquiera de la pantalla, generándose una recta, esto es, un vector. Para dibujar el perfil de una casa, por ejemplo, habrá de efectuarse esa operación varias veces. En una representación de barrido (*derecha*), del tipo de las empleadas por los televisores, el haz de electrones sigue un patrón constante de barrido en líneas horizontales; la intensidad del haz aumenta cerca de los elementos de imagen más próximos a las líneas del dibujo. Las representaciones por barrido constituyen el tipo más común de plasmación de imágenes.

más que señalarla con un lápiz óptico o con un "ratón" (un dispositivo que se desliza sobre la mesa y dirige el puntero por la pantalla). Símbolos gráficos sencillos, los llamados iconos, representan los objetos habituales en una oficina: ficheros, carpetas, papeleras, calculadoras o relojes. Y con sólo señalarlos se seleccionan las funciones que simbolizan. Se ha comprobado que la mayoría de la gente prefiere las interfases de menús e iconos a las alfanuméricas, porque, si las características de los objetos están bien diseñadas, son más naturales y resultan más fáciles de recordar y manejar; además, se cometen menos errores.

La representación gráfica por ordenador se está generalizando a otras circunstancias de la vida cotidiana. Los niños (y muchos adultos) se están "alfabetizando" en la representación gráfica practicando juegos y resolviendo ejercicios educativos que emplean efectos visuales y requieren a menudo gran destreza y participación. Muchos diseñadores producen llamativos efectos visuales y espectaculares imágenes animadas para los anuncios de televisión, y hasta efectos especiales para películas de fantasía científica. Pero consumen muchísimo tiempo de cómputo para elaborar una sola imagen de alta resolución. Trataremos más adelante de la síntesis de imágenes, uno de los campos de la representación gráfica por ordenador que registra mayor expansión.

La mayoría de las representaciones gráficas interactivas se fundan en la tecnología del tubo de rayos catódicos (si bien se han puesto de moda los paneles planos de estado sólido para diversas aplicaciones, así en ordenadores portátiles). El haz de electrones producido por esos tubos choca contra una película de fósforo adherida a la pantalla, que emite un destello de intensidad proporcional a la energía cinética de los electrones. La luz que produce el fósforo se desvanece en milisegundos, de modo que debe redibujarse la imagen constantemente, por lo común 30 o más veces por segundo. El repaso de la pantalla se efectúa según una representación digital de la imagen que se almacena en una unidad de memoria llamada tampón ("buffer") de refresco.

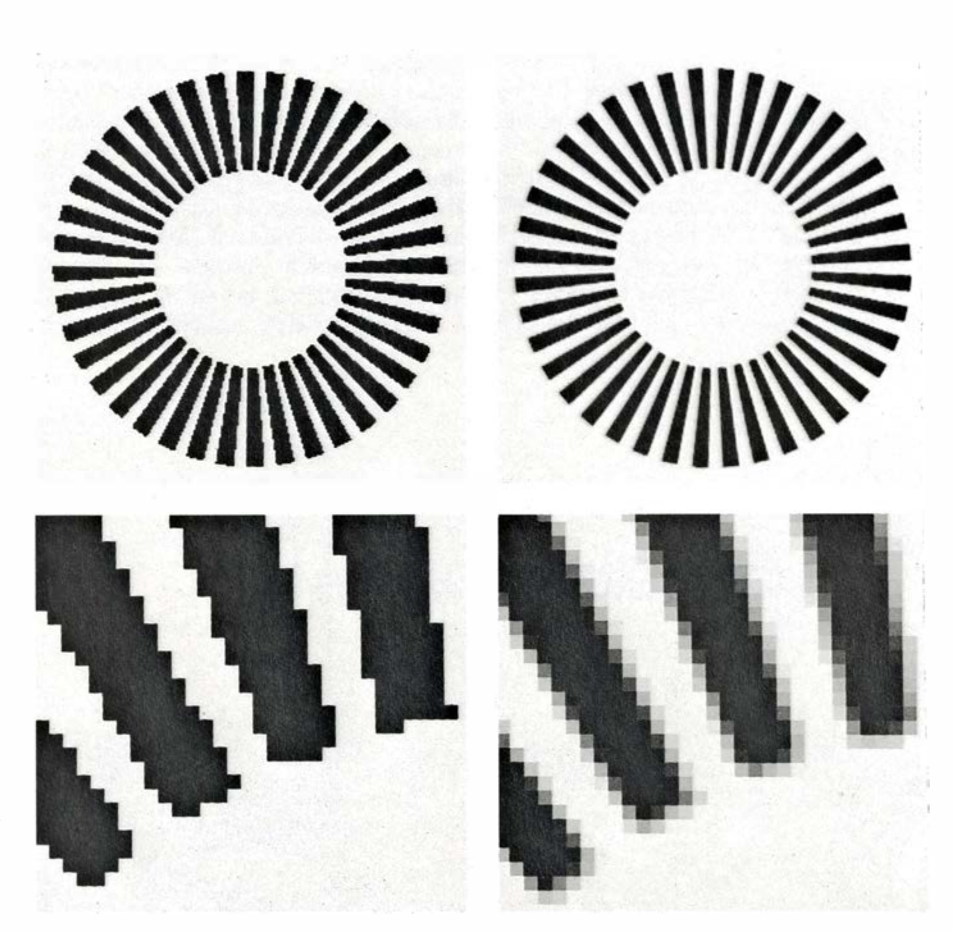
El haz de electrones puede dirigirse hacia el punto de la pantalla que interese según dos métodos: el modo vector y el modo de barrido. En el primero, el haz se deflecta continuamente entre dos puntos de la pantalla según un sistema bidimensional de coordenadas x, y; se obtiene así una línea

recta, bien definida, denominada vector. Por combinación de rectas de esas se genera un dibujo lineal. Los caracteres se componen igualmente de cortos vectores. En el registro de refresco se almacena, en forma de relación de instrucciones codificadas que especifican las coordenadas de los extremos, y otros atributos (su grosor, intensidad y color), un conjunto de imágenes elementales, las "primitivas" (líneas, arcos, caracteres y otros elementos de imagen). Los sistemas dotados de capacidad para mostrar representaciones tridimensionales en tiempo real se complementan con el soporte físico adecuado para plasmar una "transformación de la visualización", operación que consiste en la proyección de primitivas tridimensionales en la pantalla bidimensional.

Mientras la imagen se refresca, puede pedirse que el propio ordenador, u otro soporte físico al efecto, asigne valores de traslación, rotación o de escala a los valores de final de vector o a los parámetros de transformación de la visualización, para así modificar la imagen en el siguiente ciclo de refresco. Pueden especificarse esos parámetros mediante un programa de imágenes animadas, o puede hacerlo el usuario, valiéndose de un "ratón", una palanca de mando o pulsadores. Se ha demostrado que el movimiento suave de los objetos, o del punto de mira del usuario, por la pantalla tiene mucho que ver con la sensación que se experimenta: la de una retroalimentación cinestésica conforme se explora una escena tridimensional no familiar.

La representación gráfica por medio de vectores, que constituyó en un comienzo el modo usual de visualización, ofrece varias ventajas: plasma las primitivas con poco gasto de memoria; éstas se dibujan con gran precisión y el operador puede alterar continuamente la imagen en tiempo real. Su principal inconveniente es no poder rellenar las superficies; los objetos, sean bi- o tridimensionales, han de representarse como si fueran diagramas de alambre. Más aún, si el número de primitivas que hay en pantalla es tan grande que impide redibujarlas en el tiempo asignado a un sólo ciclo de refresco, se dispone de muy pocos ciclos y la imagen parpadea.

En el modo de barrido el haz no se desvía siguiendo el dibujo de la imagen a representar. Igual que en los televisores, barre toda la pantalla recreando un patrón constante. El único control se ejerce sobre la intensidad del haz. En las imágenes en color se controlan independientemente las intensidades de tres haces distintos, uno para el rojo, otro para el verde y otro para el azul. Cada haz incide sobre su correspondiente punto de fósforo de la tríada (rojo, verde, azul) que conforma cada elemento de imagen (*pixel*). Las primitivas se plasman en este modo de representación intensificando los elementos de imagen que quedan en las proximidades de las rectas, curvas o contornos definidos por los puntos extremos de las primitivas. Las áreas rellenas se obtienen intensificando todos los elementos encerrados en ellas. Los monitores de barrido suelen ser más simples y baratos que los de vector, pues su modo de deflexión es constante. Requieren, en cambio, mayor capacidad de memoria del registro de refresco, pues deben almacenar, en concepto de valor de intensidad o de color, al menos un bit para cada elemento de imagen de la pantalla. (En este caso, al registro de refresco se le llama también registro de cuadro o mapa de bits.) Una de las ventajas que ofrece el almacenamiento de la imagen en forma de elementos es que la representación no depende del número de primitivas requerido para su manifestación. Se elude con ello el problema del parpadeo.

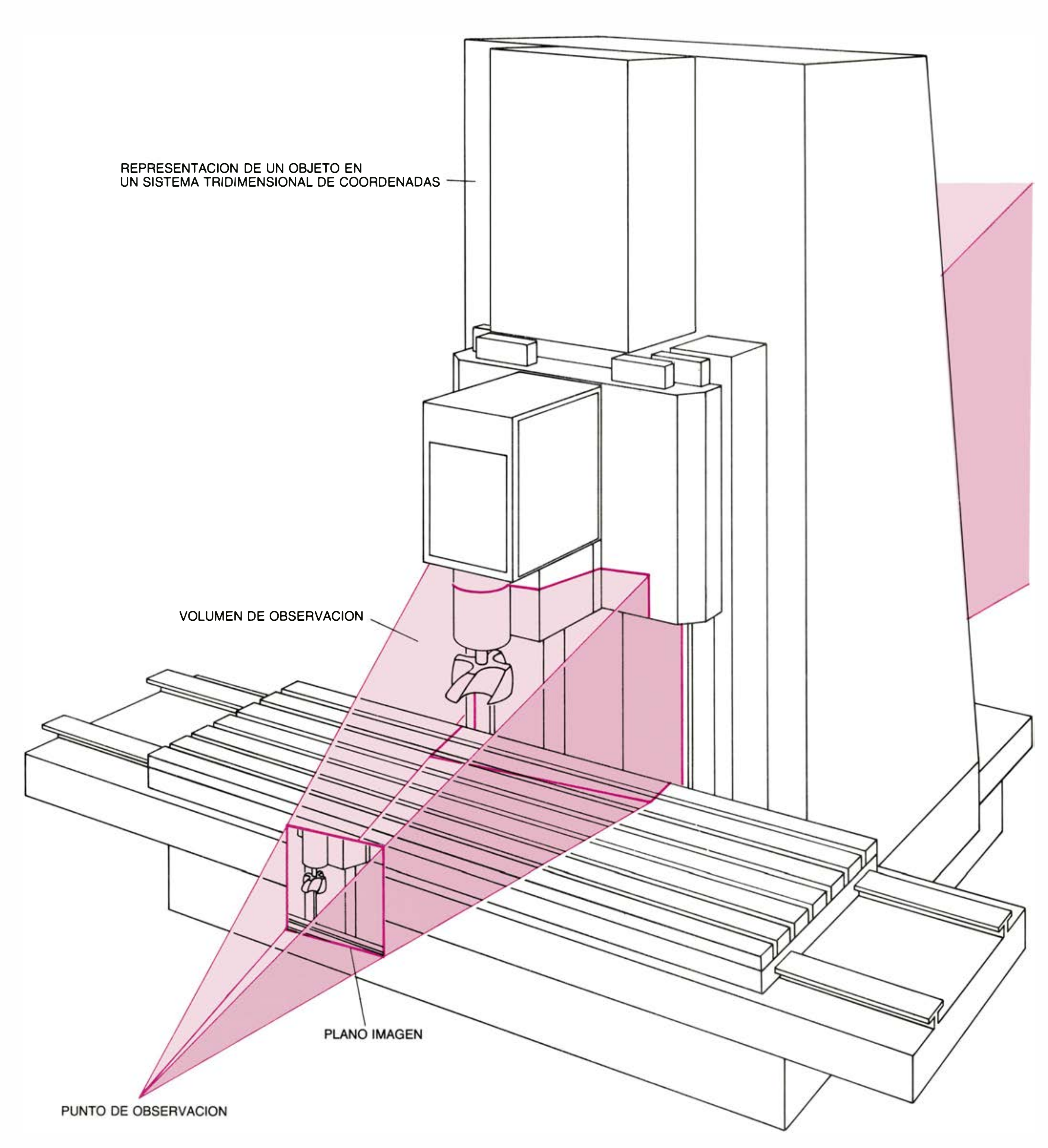


3. "DIENTES", o contornos en sierra, que aparecen en las imágenes de barrido a lo largo de cualquier línea o borde que se aparte de la horizontal o la vertical. Se producen por el modo con que los elementos de imagen más próximos plasman las "primitivas". El artefacto, denominado también escalonamiento, apenas se percibe en la figura radial izquierda superior; se aprecia mejor en la ampliación de la misma situada debajo. Se aminora en parte el defecto alterando la intensidad de los elementos que se sitúan junto a los bordes, con lo que se borra el contorno. Así se ha hecho en la imagen superior derecha y en la ampliación inferior. El problema del escalonamiento constituye un ejemplo de formación de imagenes secundarias (*aliasing*); la solución que se le da aquí se denomina de formación de anti-imagen secundaria (*anti-aliasing*). Las figuras son obra de Paul S. Strauss y James K. Rinzler, de la Universidad de Brown.

En el modo vectorial se trazan las líneas y los contornos igual que lo haría un delineante, con escuadra y cartabón. El modo de barrido, en cambio, se sirve de una versión electrónica de la técnica puntillista introducida en pintura, el siglo pasado, por el impresionista francés Georges Seurat. Esta técnica de muestreo pone a veces en evidencia los elementos de imagen, de suerte que las líneas que no son horizontales ni verticales presentan contornos en sierra. Este artefacto, denominado también escalonamiento o punteado, constituye un ejemplo de aparición de imagen secundaria (*aliasing*), problema frecuente en el procesa-

miento de señales. Puede minimizarse aumentando la resolución de la imagen o alterando la intensidad de los elementos de imagen que quedan en los bordes, de modo que se desenfoque la línea que traza el contorno. (A este último método se le ha llamado de obtención de la anti-imagen secundaria, *anti-aliasing*.) La Megatek Corporation ha desarrollado un nuevo método que simula un aumento de la resolución y evita el desenfoque. Sus diseñadores lo llaman de desfase de pixel. Consiste la técnica en desplazar vertical o lateralmente los elementos de imagen, del orden de un cuarto, la mitad o hasta tres cuartos del diámetro de los elementos. Puede también alterarse su tamaño para que rellenen los vacíos creados.

La capacidad de especificar independientemente el valor de la intensidad o color de cada elemento en los sistemas de barrido resulta de particular importancia a la hora de crear tipos de caracteres o iconos. Suele definirse un tipo por medio de matrices de elementos de imagen, una por cada carácter o icono. Cuando se precisa un carácter, o icono, se copia la matriz correspondiente, de la memoria principal del or-



4. TRANSFORMACION DE LA VISUALIZACION. Es el procedimiento por el cual la representación de un objeto (en este caso una imaginaria fresadora) se proyecta desde un sistema de coordenadas tridimensional "del mundo" en el sistema de coordenadas bidimensional que constituye la pantalla, representada por el plano imagen en este diagrama. Imitando el proceder que sigue una máquina fotográfica, el soporte lógico "atenaza" las porciones de la imagen que quedan fuera del campo de visión y proyecta únicamente las que quedan dentro. En una proyección con perspectiva tridimensional, el límite de atenazamiento suele ser una pirámide. Aquí se eliminaron los bordes ocultos antes de dar los toques previos a la proyección.

denador o del circuito especial del registro de cuadro, en la porción que contiene la representación en pantalla de ese carácter. Así, resulta mucho más lento introducir cambios en las imágenes que se sirven del modo de barrido que en las de vector, pues en éste basta con codificar las primitivas alteradas, mientras que en aquél debe renovarse un elevado número de elementos de imagen. Los más modernos sistemas de barrido permiten al programador copiar o mover con rapidez bloques rectangulares en el registro de cuadro por medio de operaciones que facilitan el avance y retroceso del texto en pantalla, la redisposición de ventanas y la creación de secuencias animadas simples. Ofrecen a veces esos sistemas una relación de representaciones de la imagen, con un rápido "barrido" de la forma codificada de la primitiva al modo de elementos, en el registro de cuadro.

La programación de apoyo de alto nivel tiene por función librar al programador de los detalles del soporte físico de bajo nivel, permitiéndole mayor concentración en los aspectos que se refieren a la aplicación de que haga uso. En los comienzos de la representación de imágenes por ordenador constituía ello un logro imposible, pues los programas de aplicación de gráficos se escribían en lenguaje ensamblador. La eficacia primaba sobre la facilidad de programación, y la posibilidad de aprovechar en un tipo de ordenador los programas concebidos para otro ni siquiera entraba en consideración. A finales de la década de 1960 y comienzos de la siguiente se inaugura la tendencia a escribir programas de representación gráfica con lenguajes de alto nivel e independientes del sistema de ordenador utilizado.

El primer soporte lógico para gráficos se diseñó imitando la estrategia de entrada-salida de los lenguajes de alto nivel. Se generaron dispositivos "virtuales" imaginarios, en correspondencia con periféricos interactivos reales, por medio de programas de bajo nivel, "conductores de dispositivos" (*device drivers*); se hacían cargo del complejo soporte físico y de las comunicaciones con el procesador central (igualmente complejas). A cada imagen virtual se le asignaba una pantalla virtual cuadrada, diseñada de modo que coincidiera con el mayor cuadrado que podía soportar la pantalla real o el sistema de dibujo (*plotter*). Se empleaba el mismo sistema unitario de coordenadas para direccionar la pantalla virtual, sin atender a las dimensiones de la pantalla real. Todas las imagenes virtuales podían disponer igualmente de dispositivos virtuales de entrada. Los periféricos no disponibles en un teclado podían simularse por medio de los que sí se tenían; podían crearse, así, teclados virtuales, botones de mando virtuales e, incluso, "ratones" virtuales.

La mayoría de los programas de entonces se desarrollaron para aplicaciones de diseño ayudado por ordenador y visualización de datos. Se ejecutaban en sistemas de representación vectorial y plasmaban imágenes derivadas de una base de datos de aplicación, el denominado modelo de aplicación. El soporte lógico para gráficos proporcionaba al programador un sistema de coordenadas de un "mundo" bidimensional o tridimensional igualmente adecuado para manejar unidades en angstrom, centímetros, kilómetros o años-luz. El sistema de coordenadas permitía al programador abstraerse de la definición de primitivas en un nivel aún más apartado del soporte físico que el sistema de coordenadas convencional de la pantalla virtual. Los programas se hacían cargo, igualmente, de la operación de transformación de la visualización, especificando el área en el sistema de coordenadas que debía aparecer en pantalla y la zona de la pantalla virtual donde debía alojarse. El programa de visualización de la transformación "atenazaba" las primitivas que quedaban fuera del área representada; proyectaba en la pantalla real únicamente las primitivas que entraban en la porción visualizada. En representaciones bidimensionales, el límite de "atenazamiento" era un rectángulo; en las tridimensionales podía ser un paralelepípedo (proyecciones paralelas) o una pirámide (proyecciones con perspectiva).

Así, la primera programación de representaciones gráficas, desarrollada hace aproximadamente una década, puede compararse a una "cámara sintética": el programa de aplicación construye un universo de objetos, verbigracia, símbolos de organigramas, elementos de circuitería o átomos, ajustándose al modelo que dicte la aplicación e incorporando todos los atributos y parámetros apropiados; extrae luego la información geométrica que trasladará a los programas de representación gráfica. Estos, que suelen quedar bajo control del usuario, toman una "instantánea" de las primitivas especificadas en el universo del espectador desde el punto de mira elegido y la transfieren a la pantalla. De este modo, al programa de aplicación le corresponde la responsabilidad del modelado y, la de la visualización de las partes, al programa de "cámara sintética". La propia aplicación suele constar de dos subsistemas: un editor de gráficos que pemite al usuario crear y manipular el modelo de aplicación y su representación visual, y un juego independiente de paquetes de postprocesado que analizan el modelo terminado. En el diseño ayudado por ordenador, esos paquetes aportan lo necesario para simular y ensayar el diseño, así como para especificar valores de uso en la manufactura y construcción, que a menudo corren a cargo de máquinas controladas numéricamente.

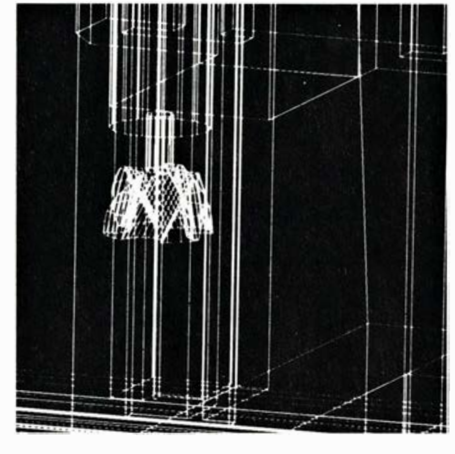
Contamos ya con dos paquetes normalizados de representación gráfica para todas las categorías de monitores comerciales: el Core Graphics System, tridimensional, subvencionado por la Asociación norteamericana de Maquinaria de Cómputo, y el Graphical Kernel System, bidimensional, adoptado por la Organización Internacional de Pesos y Medidas. Ambos derivan de un antepasado común, y son, en esencia, paquetes de cámaras sintéticas. Se diseñaron antes de que se popularizara el modo de barrido. Aunque pueden trabajar con primitivas de este modo (matrices de elementos de imagen, polígonos rellenos, etcétera), operan con el sistemas de coordenadas del mundo, es decir, con objetos definidos por el usuario. En muchas aplicaciones simples de los gráficos de barrido, el programa no obtiene, de las posibilidades del paquete, un provecho que justifique la hipertrofia de cálculo requerida al tratar aplicaciones más complejas. Es más, un programa que se apoye en un paquete de gráficos bien pudiera no aprovechar adecuadamente las nuevas y poderosas prestaciones del soporte físico incorporadas en las representaciones gráficas y asociadas a los terminales gráficos de barrido y ordenadores personales.

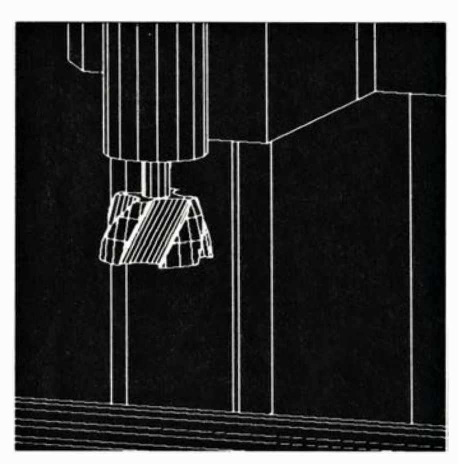
Se están popularizando, en sistemas de barrido, programas que no se ajustan al esquema de modelado-seguido-de-visualización propio de la "cámara sintética" y que admiten la opción de "pintar". Lo que se manipula en esos programas no son objetos en coordenadas del mundo, sino los propios elementos de imagen; los paquetes de programación permiten al usuario alterar el color, mover y combinar lógicamente regiones arbitrarias del registro de cuadro. Pintar manipulando los pixeles del registro de cuadro viene a ser como sacar una fotografía alternado directamente zonas de la emulsión en vez de exponer la película a una escena real a través de una cámara. Los paquetes de

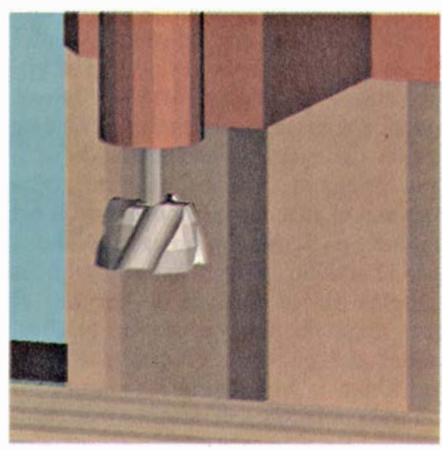
gráficos del tipo "cámara sintética" no son adecuados para esas operaciones de bajo nivel, ligadas a periféricos.

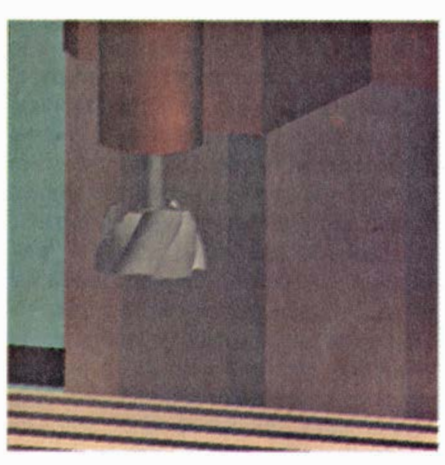
La situación se complica más aún por la necesidad de manejar a la vez varias ventanas en una pantalla cartografiada en bits. Los paquetes de gráficos en uso son incapaces de atender a más de una aplicación. De ahí que la mayoría de los terminales gráficos lleven incorporado un "gestionador de ventanas", una porción de bajo nivel del soporte lógico que sigue la pista a los programas que se ejecutan en cada ventana y a la ubicación de las ventanas sobre la pantalla. Debe lograrse que en una ventana se ejecute un programa de dibujo, en otra el tratamiento de un texto y en una tercera un programa de aplicación que se sirva de paquetes convencionales de representación gráfica. El gestionador de ventanas debe resolver problemas como el desplazamiento de las ventanas sobre la pantalla, la sobreposición y solapamiento de ellas, el escalado y atenazamiento de las primitivas para que se ajusten a la porción visible de una ventana y el barrido de las primitivas para su aparición en pantalla. No se dispone todavía de ningún diseño de esos gestionadores que convenza a todo el mundo.

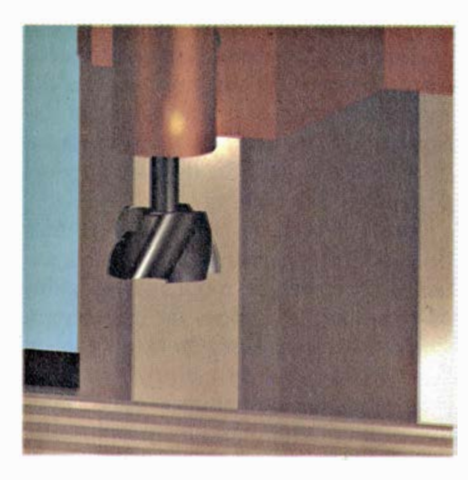
En un futuro próximo coexistirán varias normativas sobre representación gráfica diseñadas por distintos grupos de usuarios. En América del Norte, la Initial Graphics Exchange Specification dicta normas de representaciones de ingeniería en el diseño dirigido por ordenador; la North American Presentation Level Protocol Syntax hace lo propio para la exhibición de textos e imágenes en televisión. Todas las reglas coinciden en su finalidad: definir primitivas y sus atributos y agruparlas en conjuntos rotulados que permitan su manejo selectivo en bloque. En última instancia, todas ellas habrán de unificarse.













5. SINTESIS DE IMAGENES POR ORDENADOR. Cabe imaginarla como una sucesión de fases, si bien en los programas reales suelen éstas imbricarse. En este ejemplo, el objeto (una ampliación de la fresadora) se define por medio de una malla de polígonos; se representa en primer lugar a modo de diagrama de "alambres" (*1*). Se eliminan luego los bordes que quedan ocultos (*2*). En el paso siguiente se aplica un matizado (en este caso de color) a cada uno de los polígonos; al efecto se tiene en cuenta el ángulo que forma cada polígono con la fuente de luz, y sus cualidades superficiales. Se obtiene entonces una imagen poco verosímil (*3*). Pueden suavizarse las discontinuidades de los lados comunes a dos polígonos aplicando el matizado de Gouraud (*4*); se añaden reflejos con el matizado de Phong (*5*). En el último paso se suaviza el dentado (*6*). Las imágenes las obtuvieron Rinzler y Strauss, en colaboración con Roger L. Gould, Richard L. Hagy, David H. Laidlaw y Gerald I. Weil, todos alumnos de Brown.

En su mayoría, las aplicaciones tradicionales de las representaciones gráficas por ordenador han sido bidimensionales. Ultimamente, sin embargo, se aprecia un creciente interés comercial por las aplicaciones tridimensionales, en especial a raíz de los avances registrados en la última década hacia la resolución de dos problemas gemelos: el modelado de escenas tridimensionales y su representación con la máxima verosimilitud. En los simuladores de vuelo para entrenamiento de pilotos, por ejemplo, el interés reside en dar respuesta a las entradas que parten tanto del piloto como del instructor. Para dar impresión de movimientos suaves, el simulador debe presentar una imagen bastante realista de un mundo en cambio dinámico a un ritmo no inferior a los 30 cuadros por segundo. Las imágenes elaboradas con fines publicitarios o por empresas de espectáculos, en cambio, no se obtienen en tiempo real, sino fuera de línea, y, para alcanzar un realismo o impacto visual máximos, suelen requerir tiempos de procesamiento de varias horas. En el diseño dirigido por ordenador crece el interés por los sistemas interactivos que, a partir de un patrón de "alambres" visualice acto seguido una versión acabada de los mismos. El más reciente soporte físico permite, incluso,

la creación interactiva de objetos poligonales "macizos".

Los objetos bidimensionales se diseñan valiéndose de ciertas primitivas: rectas delimitadas por dos puntos extremos; polígonos definidos por una relación de vértices y probablemente un modelo de relleno; círculos definidos por un centro, un radio y quizá también un modelo de relleno y curvas polinómicas, definidas por sus coeficientes. En el espacio tridimensional, las primitivas correspondientes se obtienen añadiendo la coordenada z. Cabe también definir primitivas que existan sólo en tres dimensiones: poliedros, pirámides, esferas, cilindros y superficies descritas por ciertas funciones polinómicas.

Los sistemas de modelado de objetos sólidos tridimensionales se apoyan en la especificación interactiva, o bien fuera de línea, de los parámetros. La especificación fuera de línea se logra mediante archivos de datos creados por otro programa, o sirviéndose de un editor de texto. También puede efectuar ese trabajo una descripción de procedimiento, del tipo de las que se emplean para generar curvas fractales y paisajes. Es más, los objetos pueden modelarse, ya directamente, como un sólido geométrico, ya indirectamente, como volúmenes limitados por sus superficies.

En los sistemas que emplean la geometría constructiva de sólidos, los objetos se modelan directamente por medio de primitivas macizas, bloques, cilindros y esferas, por ejemplo. Pueden combinarse las primitivas valiéndose de juegos de operaciones tridimensionales, como la unión (el empalme de dos objetos), la intersección (la elección de un subconjunto común) y la diferencia (la elección de todas las partes del primer objeto que no tenga en común con el segundo). Se acude a la representación indirecta en los sistemas de representación limítrofe que, si bien ofrecen operadores del conjunto, definen los objetos como si les rodearan lados poligonales, cilíndricos e incluso retazos de superficies definidas por funciones polinómicas. Tal definición de superficies "de forma libre" por medio de segmentos curvos desempeña actualmente una importante labor en el diseño de vehículos aeroespaciales y automóviles.

Un objeto que presente simetría rotacional puede igualmente describirse por medio de una superficie de revolución; un vaso, o una botella, se definen por su generador (el perfil de la silueta) y un eje de revolución. Aná-



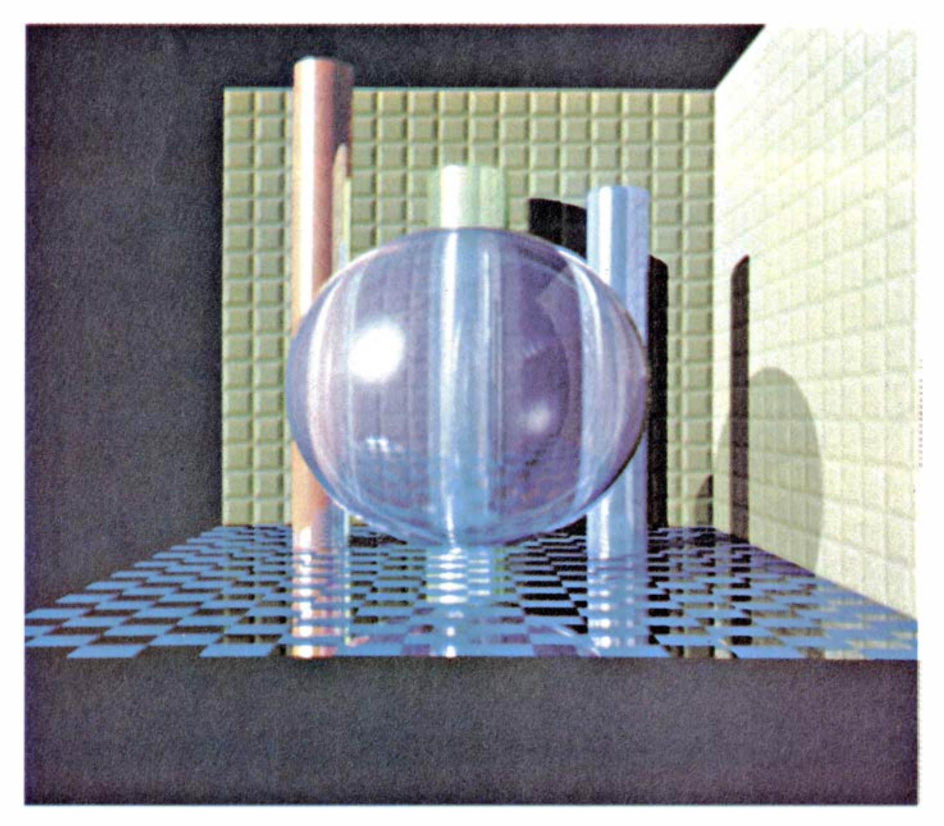


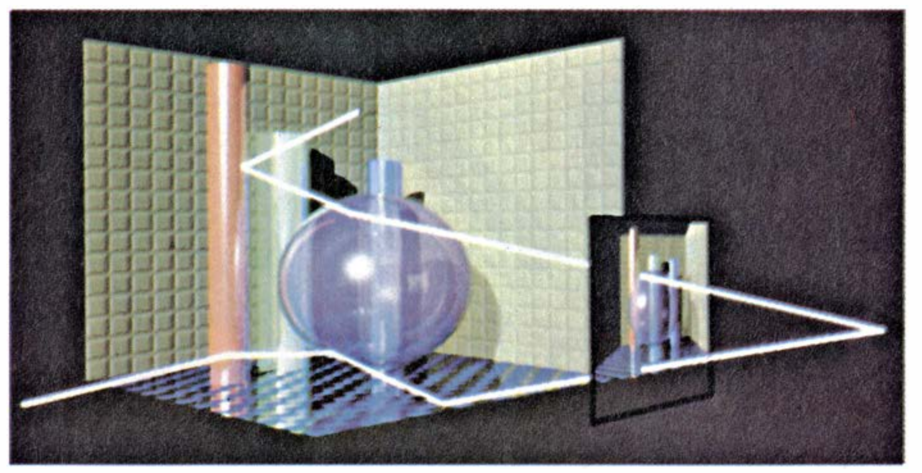


6. ANIMACION DE GRAN CALIDAD A TIEMPO REAL. Se alcanza en ciertas aplicaciones específicas, como esta secuencia seleccionada de un sistema de simulación de vuelo para entrenamiento de pilotos en las maniobras de aprovisionamiento de combustible en el aire. Este sistema de prestaciones ultrarrefinadas, capaz de generar 50 cuadros por segundo, es de la compañía Evans & Sutherland.

loga a la generación por rotación es la que emplea la traslación; en este caso, para crear un volumen se traslada, a lo largo de una curva espacial, una superficie de complejidad arbitraria (puede, por ejemplo, presentar huecos). Se obtiene un engranaje imaginario definiendo una cuarta parte de su plano frontal; valiéndonos de operaciones de simetría se completa éste y, por traslación en línea recta, se obtiene la forma cilíndrica del engranaje macizo.

Muchas otras técnicas matemáticas son de utilidad a la hora de definir clases de objetos; los sistema híbridos se valen de varias. El caso especial del diseño de objetos por medio de métodos interactivos presenta un problema adicional: se fuerza al usuario a observar una proyección bidimensional de una escena tridimensional; en esa situación, resulta difícil valorar la profundidad. Entre las técnicas que facilitan el camino al usuario a medida que avanza el proceso de especificación se cuentan las vistas múltiples (así las proyecciones ortogonales habituales en planta, alzado y costado, o las vistas en perspectiva tridimensional) o el dibujo en un plano de las coordenadas x, y o z constante. Otras ayudas son la actualización dinámica de las líneas de dimensión o las redes bidimensionales o tridimensionales con marcas de señalización.





7. TECNICA DE TRAZADO DE HACES. Aprovecha un algoritmo muy lento que calcula la reflexión y la refracción de la luz en la superficie de objetos imaginarios. El programa traza los distintos rayos luminosos, uno a uno, desde el punto de observación y retrocediendo por todos los pixeles del plano imagen hasta que los rayos dan contra una superficie. Se continúa el trazo del haz reflejado para decidir si procede de una fuente de luz, ya sea directamente o bien tras su reflexión en otro objeto. Si se trata de una superficie transparente debe reseguirse otro haz, el de refracción. En este ejemplo de aplicación de la técnica, obra de Lee Westover y Turner Whitted, de Carolina del Norte, se muestra arriba una imagen de trazado de haces y, en el diagrama generado por ordenador de abajo, el procedimiento seguido para obtenerla. Las líneas blancas resiguen dos haces reflejados; se han omitido las componentes refractadas.

Tras la definición de los objetos de la escena, se transfieren sus descripciones a los programas de síntesis de imagen para su presentación. Los actuales algoritmos de síntesis de imágenes operan con descripciones poligonales y polinómicas, o con otras definiciones de superficies matemáticas de rango superior. Antes de la presentación es común reducir las definiciones de alto nivel a aproximaciones simplificadas compuestas por una red de polígonos menores. Cabe imaginar el proceso de presentación como una serie de pasos que, en los programas reales, se dan entrelazados. Todos son, en esencia, adaptaciones de las leyes fundamentales de la óptica.

El primer paso consiste en eliminar cualquier superficie tapada, es decir, las caras, o porciones de caras, que no quedan a la vista del punto de mira de la cámara sintética. Se encuadran en esta categoría tanto las superficies de la cara oculta de los objetos como las que tapan otros objetos situados más cerca del observador. Pueden incorporarse al soporte físico diversas técnicas que lo lleven a efecto. El algoritmo parte del principio de que la pantalla se ubica en el plano de proyección $z = 0$, con la escena detrás. Por ejemplo, el algoritmo del registro z mantiene, en un registro independiente, los valores de z de cada elemento de imagen; un valor para cada pixel. El valor z de cada pixel registra la profundidad del punto correspondiente del polígono que se halle más próximo y se proyecte en ese elemento de imagen. Cuando otro polígono se transforma, se atenaza y se proyecta en el plano $z = 0$, se comparan uno por uno los valores de z de sus elementos componentes con los almacenados en el registro z. Sólo si el valor de z de un elemento del polígono actual es inferior (señal de que se halla más próximo a la pantalla en ese elemento que cualquier polígono anterior) se proyectará ese elemento de imagen y se almacenará, en los registros de refresco y de z. En realidad se visualizará siempre que no lo remplacen al procesarse el último polígono.

Mientras que el algoritmo del registro z toma los polígonos en cualquier orden, el "algoritmo del pintor" los distribuye, desde el más alejado hasta el más cercano. Si un par de polígonos no pueden ordenarse, se subdividen hasta

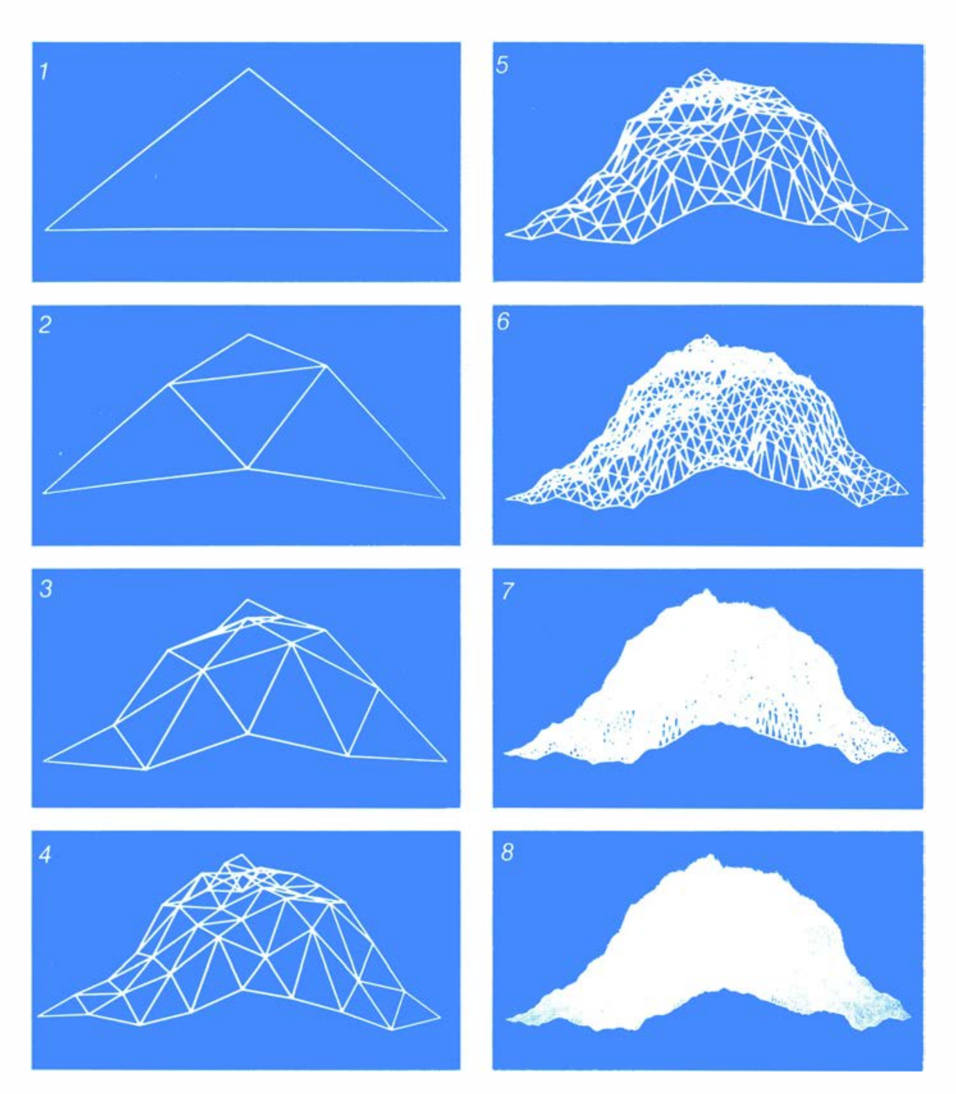
que sus partes sí puedan. Seguidamente se proyectan y se "pintan" en el registro de cuadro, de atrás adelante. Así, los más cercanos eclipsan a los más alejados, sin más necesidad de cómputo.

Tras el cálculo de las superficies visibles se procede a su matizado. La regla de matizado debe tomar en consideración las propiedades de la superficie (su color, textura y reflectancia), así como la ubicación relativa, orientación y propiedades de las fuentes de luz y otras superficies. Los modelos de iluminación distinguen luz ambiente, fuentes puntuales (el sol o una bombilla de incandescencia) o fuentes difusas (así una ventana o una fila de tubos fluorescentes).

La manera más simple de modelar la luz consiste en agregar una cantidad constante de intensidad lumínica a todas las superficies. Esa estrategia, obviamente, no distingue calidades de superficies. La reflexión de las fuentes puntuales de luz en superficies mates, o difusas (las que dispersan la luz por igual en todas direcciones) se modela por medio de la ley del coseno de Lambert, que sostiene que la intensidad varía según el coseno del ángulo que forma la dirección de la fuente de luz con un vector perpendicular a la superficie, el llamado vector normal. La iluminación más intensa se produce cuando la fuente es perpendicular a la superficie. En las superficies muy brillantes (especulares), como la madera bien encerada o los metales, la cantidad de luz reflejada depende de los ángulos que forman con el vector normal la fuente de luz y el punto de mira del observador. La superficie se comporta como un espejo, en cuanto que refleja casi toda la luz cuando esos ángulos son iguales, a saber, cuando el punto de mira y la fuente de luz se disponen simétricamente respecto de la normal a la superficie. A medida que los ángulos se apartan de esa simetría, la intensidad de la luz disminuye rápidamente. La intensidad de cada supeficie se computa sumando sus componentes de reflexión ambiental, difusa y especular. Si la imagen es en color, se emplea una ecuación para cada uno de los tres primarios.

El efecto resultante de esas operaciones es una imagen muy falsa, segmentada en facetas. Puesto que los polígonos se describen a partir de un solo vector normal, a los polígonos adyacentes que posean normales distintas les corresponderán valores de intensidad diferentes, con lo que se produciría una destacada sensación de discontinuidad en los bordes comunes. El sombreado de Gouraud (ideado por Henri Gouraud) promedia las intensidades en los vértices de los polígonos, y a lo largo de las líneas de barrido, para obtener un efecto más matizado. El sombreado de Phong (que diseñó el fallecido Bui-Tuong Phong) mejora al anterior: se sirve de un cálculo más elaborado y sensible a los efectos direccionales de brillos especulares. Los más modernos sistemas de barrido de grandes prestaciones y precio medio presentan unos 3000 polígonos por segundo. Comienza el cómputo elaborándose una jerarquía de objetos, incluida la aplicación de transformaciones geométricas a sus componentes para simular el movimiento; sigue con el cálculo de la transformación de visualización y la aplicación de los algoritmos de eliminación de las superficies ocultas y de matizado. Hace pocos años, esas prestaciones sólo las ofrecían los simuladores de vuelo altamente especializados y extraordinariamente caros.

Otros factores a tener en cuenta en la representación de imágenes son las sombras, la transmisión de luz y las propiedades de las superficies, como su textura y grano. Los algoritmos de sombreado para fuentes de luz puntuales se asemejan a los de ocultación de superficies: determinan qué superficies quedan a la "vista" de la fuente de luz y del observador. Las superficies que se ven a la vez desde ambas direcciones no caen en la sombra; las que ve el obser-



8. TECNICA SIMPLE de obtención de una imagen verosímil de una montaña. Se basa en última instancia en los conceptos de geometría fractal formulados por Benoit B. Mandelbrot, del Centro de Investigación Thomas J. Watson de la IBM. Este ejemplo se reproduce aquí por cortesía de Lucasfilm Ltd. Partiendo del triángulo que se muestra en *1*, el programa genera el paso *2* de acuerdo con el siguiente procedimiento: primero divide cada lado del triángulo por su punto medio. A continuación desplaza los puntos medios en una distancia proporcional a la longitud del lado correspondiente. (El factor de proporcionalidad puede generarse estocásticamente o tomarse de una tabla de, por ejemplo, 100 números aleatorios bien dispersos.) En tercer lugar, conecta los tres nuevos puntos entre sí formando cuatro nuevos triángulos. El paso *3* se obtiene del *2* aplicando el mismo procedimiento a cada uno de los triángulos, con lo que se generan 16 triángulos, a los que, a su vez, se les aplicará esa rutina en el paso *4*; igual se procede en los pasos siguientes. Si bien el algoritmo de subdivisión es simple, puede generar superficies poligonales notablemente complejas. La superficie con aspecto de montaña del paso *8* puede, a continuación, plasmarse en un paisaje acabado por medio de las técnicas convencionales de tratamiento de imágenes.



9. IMAGEN COMPUESTA de un paisaje costero, titulado "Punta Reyes". La ha diseñado un equipo de Lucasfilm. En su obtención se han empleado diversas técnicas; los distintos elementos del paisaje se elaboraron por separado y a continuación se combinaron. Loren Carpenter se sirvió de la sencilla técnica de modelado expuesta en la figura 8 para definir las rocas, montañas y agua; escribió asimismo el programa de ocultación de superficies y un programa de "atmósfera" para el cielo y la neblina. Rob Cook dirigió el proyecto, diseñó la carretera, las colinas, la valla y el arco iris; redactó también el programa de generación de texturas. Tom Porter contribuyó con el procedimiento de dibujo de la textura de las montañas y escribió los programas para combinar los elementos en una imagen compuesta. Bill Reeves definió la hierba sirviéndose de un sistema de "partícula móvil" de su propia invención; también escribió el soporte lógico de modelado. David Salesin puso las olas en las lagunas y, Alvyn Ray Smith, las flores.

vador, pero no la fuente de luz, sí. En los complejos cálculos de las luces difusas debe considerarse la sombra y la penumbra.

La transmisión de luz constituye un tema más complejo aún. La "especular", característica de las superficies transparentes, como el vidrio, se determina a partir del índice de refracción de la materia. La transmisión difusa a través de materiales traslúcidos, como el vidrio esmerilado, provoca la dispersión en todas direcciones. Los algoritmos más complejos y que rinden efectos más reales son los denominados trazadores de haces. En esencia, resiguen los distintos rayos para determinar cuáles acaban en el punto de mira y cómo han llegado hasta él. Para eludir el tratamiento de una infinitud de haces, el proceso opera a la inversa, esto es, comienza en cada elemento de imagen. Todos los rayos que parten del punto de mira y atraviesan un elemento de imagen se proyectan hacia atrás, hasta que inciden en la superficie. Se resigue la trayectoria inversa de cada rayo reflejado hasta determinar si procede de una fuente de luz o de la reflexión en otro objeto. En caso de que se trate de una superficie transparente, debe trazarse otro rayo, el de refracción. Cada rayo constituye una sonda, de la que debe analizarse su intersección con todos los objetos. Sólo una pequeña fracción de los rayos proceden, en última instancia, de una fuente lumínica. Se dispone ya de nuevos métodos de tratamiento de la reflexión y la refracción de la luz difusa, pero resultan extraordinariamente costosos en términos de tiempo de cómputo.

La textura superficial puede abordarse por diversos métodos de generación de irregularidades superficiales. Para dibujar un esquema bidimensional sobre una superficie podemos recurrir a un patrón de valores de intensidad que module las intensidades computadas por los algoritmos de matizado y sombreado. Los trabajos más recientes sobre síntesis de imágenes estudian efectos visuales como la profundidad de campo, la estela de los objetos en movimiento y la representación realista de objetos naturales que exhiben regularidades e irregularidades estadísticas, verbigracia, las montañas, el agua, el cielo, los árboles y los arbustos.

Trabajemos con simples diagramas de bloques o con imágenes de gran verosimilitud, la función primordial de la representación gráfica por ordenador será aumentar nuestra comprensión, permitirnos la experimentación sin peligro, incomodidad ni excesivo costo y facilitar la respuesta a cuestiones del tipo "¿qué pasaría si...?" En la mayoría de los estudios sobre creación de modelos y de simulación, no bastarán las meras representaciones estáticas: los fenómenos que suelen interesarnos son dinámicos. Quizás una imagen fija valga por mil palabras, pero sin duda una imagen en movimiento vale por muchas estáticas. Una capacidad esencial de la nueva generación de potentes ordenadores es que plasman el comportamiento de los objetos a medida que cambian en el tiempo, valiéndose de animación a tiempo real.

Entre los objetos con comportamiento dinámico que interesan a los programadores se cuentan los programas y sus correspondientes estructuras de datos. Desde que se iniciara el empleo de representaciones gráficas por ordenador, allá por los primeros años de la década de los 60, se viene buscando un enfoque diagramático del diseño de los soportes físico y lógico. Son de uso ya los diagramas de bloque, curso de la información, interconexión modular, de flujo de datos y otros muchos símbolos en sustitución del tecleado de sus enunciados. Si bien gran parte del diseño del soporte físico se apoya ya en símbolos gráficos, no existe aún lenguaje de programación en el cual los elementos fundamentales sean pictóricos y no literales.

Un lenguaje común de programación, como FORTRAN o Pascal, acude a los paquetes de programas para gráficos mediante "llamadas", en vez de valerse para ello de especificaciones gráficas. La razón de tan curiosa discrepancia entre las técnicas de especificación empleadas para el soporte físico y el lógico reside en la compacidad y precisión que ofrecen los lenguajes de programación convencionales y en la facilidad con que pueden alterarse por medio de editores de texto que aporten las prestaciones adecuadas para la búsqueda de cadenas de caracteres. Se carece aún de la correspondiente experiencia con lenguajes pictóricos.

Si bien los especialistas en informática conocen bien los elementos de que se valen los programadores para especificar *cómo* efectuar una tarea en un lenguaje de programación convencional, no han resuelto todavía el problema de que el usuario declare *qué* pretende hacer y que el ordenador compile automáticamente un procedimiento adecuado que lo lleve a cabo. Sí se ha avanzado considerablemente en el diseño de "ambientes de programación" basados en las terminales de trabajo (*work-stations*). En general, esas instalaciones permiten al programador editar interactivamente y "de-

purar" los programas ayudados de presentaciones múltiples de programas y datos, plasmados en forma de texto o iconos. Las representaciones se actualizan a medida que se ejecuta el programa.

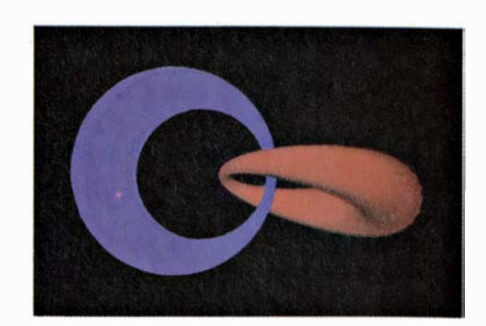
Las aulas y los laboratorios prometen constituir centros especialmente adecuados para la aplicación de representaciones gráficas por ordenador. Son numerosas las universidades y escuelas que están adaptándose a la enseñanza con microordenadores. Uno de los métodos adoptados es el tradicional "aprendizaje programado" asistido por ordenador, que pone especial énfasis en la adquisición de conocimientos y destreza. La enseñanza con ordenadores se enriquece ahora con la simulación de experimentos de "laboratorio" y el "hojeado", que permite el acceso a la información siguiendo el modelo de las enciclopedias y bibliotecas. Hace cinco años, a instancias de mi colega Robert Sedgewick, el departamento de informática de la Universidad de Brown inició el estudio de la aplicación de la tecnología de las terminales de trabajo a la educación y la investigación. El año pasado inauguramos un aula electrónica dotada de 55 terminales de trabajo de alto rendimiento conectadas a una red de gran velocidad. La mayoría de los cursos de introducción a la informática se imparten ya en ese auditorio, así como algunas lecciones de los cursos sobre ecuaciones diferenciales, geometría diferencial y neurología.

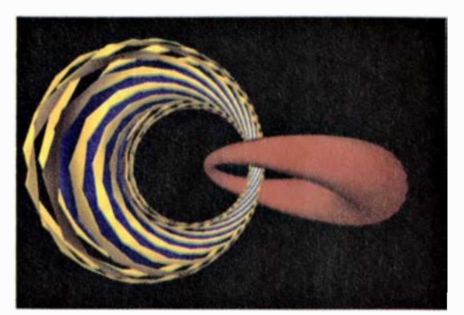
Nuestro propósito es ofrecer a los alumnos la posibilidad de "ver" un fenómeno abstracto, y con ello fomentar el desarrollo de cierta intuición geométrica, para introducirse luego en los detalles de la programación o la matemática. Pretendemos igualmente motivarles más deprisa hacia la materia, combinando la clase dictada tradicional con la experimentación de laboratorio. La mayoría de los alumnos muestran, al impartírseles las enseñanzas según el método tradicional, una actitud pasiva, aún si se les anima a plantear preguntas. No se enfrentan, por tanto, a la materia dictada hasta que abordan los deberes o los ejercicios de laboratorio. El instructor puede ahora introducir una nueva materia valiéndose de una secuencia animada de imágenes que se les presentará a todos los alumnos, y con la que deberán trabajar todos ellos de forma independiente e interactiva.

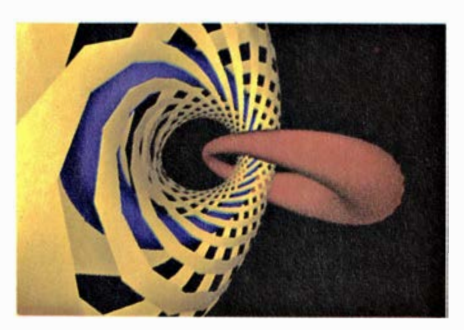
Nuestro instrumento principal, el Brown Algorithm Simulator and Animator, BALSA, (Algoritmo Brown de Simulación y Animación), permite al usuario controlar la velocidad de una secuencia animada, decidir el ángulo de visión a un objeto y especificar los datos de entrada que deban procesarse. Se puede ejecutar los programas al revés y deshacer sus efectos gráficos.

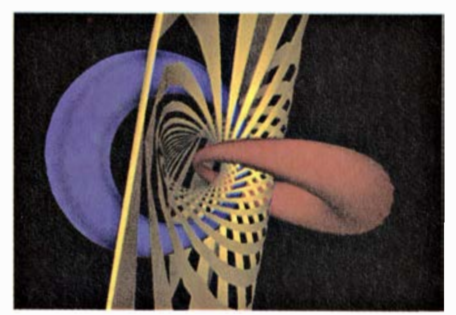
La visión dinámica múltiple de objetos complejos se ha demostrado de gran valor para los investigadores, no ya sólo para los alumnos. Se está trabajando a fondo en Brown para lograr la extensión de esa técnica otros campos de la ciencia y la matemática, así como hasta ámbitos donde no se suelen representar los objetos y procesos de estudio. ¿Existen razones intrínsecas para explicar la ausencia de representaciones pictóricas en determinadas materias, o se trata simplemente de un artefacto cultural? Si se han encontrado múltiples formas de servirse de los gráficos en distintas disciplinas, ¿qué haremos para que se incorporen a la pedagogía de la clase? La cuestión gana trascendencia con la proliferación de terminales de trabajo en los centros universitarios y el acceso ininterrumpido que los estudiantes tienen a ellas, hasta en sus dormitorios. ¿Cómo lograrán los profesores no habituados a la programación especificar y realizar las materias de su curso eludiendo el tiempo habitualmente requerido para la preparación de las clases dirigidas por ordenador, unas 100 horas o más por hora de clase? Se precisarán muchos años de experimentación y desarrollo hasta la difusión generalizada de esas prometedoras prestaciones.

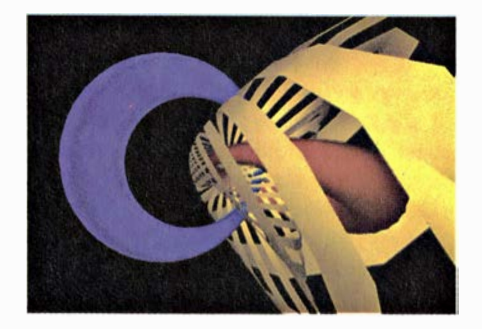
Las demostraciones que efectuamos en nuestra aula de Brown valen de













10. APLICACION MATEMATICA de la representación gráfica por ordenador, ejemplificada por una secuencia de imágenes adaptadas de la película animada "Topology and Mechanics", de Huseyin Kocak, Frederick Bisshopp, Thomas F. Banchoff y David Laidlaw, de la Universidad de Brown. Puede visualizarse una hiperesfera, el equivalente en cuatro dimensiones a una esfera, "rellenándola" con dos círculos entrelazados y una serie de superficies toroidales circundantes. (La operación es más o menos equivalente a cortar una esfera por dos puntos opuestos e intercalar una serie de círculos paralelos.) La secuencia muestra una sucesión de proyecciones en perspectiva sobre el espacio tridimensional, efectuadas desde un punto de observación de la hiperesfera, de dos superficies toroidales (una azul y otra roja), que envuelven apretadamente los dos círculos de la hiperesfera. Una tercera superficie toroidal (en amarillo) se muestra desplazándose desde la superficie azul hasta la roja, en seis pasos. Se ha cortado en tiras la superficie amarilla para poner de manifiesto su engarce con las otras dos. Este método de descomposición de las hiperesferas se inspira en la obra del matemático alemán Heinz Hopf. El computador ofrece una ayuda inestimable a la hora de plasmar esos procedimientos matemáticos, pues con gran facilidad se logra que manipule objetos abstractos de dimensiones espaciales superiores.



11. AULA ELECTRONICA diseñada por Robert Sedgewick y el autor, junto con sus colegas del departamento de informática de la Universidad de Brown. Está dotada de 55 terminales de trabajo (*work-stations*) de altas prestaciones enlazadas en una red de gran velocidad. El profesor presenta en primer lugar un tema por pantalla, valiéndose de una secuencia animada que observan todos los alumnos. A continuación, todos ellos trabajan de manera independiente sobre la misma "película interactiva". El aula electrónica se ha empleado ya para impartir cursos de informática, matemáticas y neurología.

ejemplo de la categoría, más general, de lo que podría denominarse "libros electrónicos". Otro ejemplo, algo distinto, lo constituye el Spacial Data Management System (Sistema de gestión de datos espaciales), desarrollado en el Instituto de Tecnología de Massachusetts, MIT. El cursor permite hojear un "terreno de datos" bidimensional, un escritorio de dimensiones arbitrarias poblado de iconos. Se desplaza el cursor sobre cualquier icono para que éste muestre su contenido. El banco de datos almacena textos, diagramas, fotografías, sonido e imágenes de televisión (recuperadas en tiempo real).

En un sistema experimental emparentado, la metáfora del escritorio se sustituye por una red asociativa de páginas de texto y notas al pie, al margen y remisiones. El lector puede reseguir un trazado de remisiones entre temas relacionados, como lo haría en una enciclopedia. Theodor H. Nelson, líder del movimiento en favor de la explotación de la tecnología de representación por ordenador en la creación de nuevas formas literarias, ha denominado hipertexto a ese tipo de manuscrito no lineal. Hay otros proyectos; en la frontera de la representación de imágenes se cuentan los elaborados por P. Negroponte, y colaboradores, del MIT: un manual de reparación de automóviles, interactivo, que, para explicar el texto, elabora imágenes ad hoc a partir de secuencias de fotogramas almacenados en un videodisco, y un periódico electrónico, que repasa constantemente los informes de agencias y sus bases de datos, y elabora noticias e imágenes a partir de un perfil de interés del lector.

Los profesionales de la informática, en particular los especialistas en representación gráfica, se felicitarán porque, finalmente, empiezan a cumplirse nuestras promesas acerca de la utilidad y conveniencia de los ordenadores. Las representaciones por ordenador, antes dominio de expertos, son hoy algo habitual incluso para los alumnos de la escuelas elementales, que manejan ventanas, ratones e imágenes de ordenador para dibujar y, desde luego, para desarrollar su imaginación. Reflexionar y programar valiéndose de gráficos constituyen cada vez más una parte del aprendizaje de algoritmos.

¿Qué nos depara el futuro? Para que el usuario medio disponga del equivalente a un simulador de vuelo deberán producirse mejoras de varios órdenes de magnitud en la relación precio/prestaciones del soporte físico; también habrá de avanzarse considerablemente en la comprensión de la interacción de las leyes de la física y la estética para que los artistas de la representación por ordenador logren plasmar escenas inequívocamente reales y que, además, agraden a la vista. En última instancia, las imágenes verdaderamente tridimensionales nos adentrarán en otro terreno: la investigación de hologramas digitales, que quizá lleve a la plasmación de escenas "vivas" en tiempo real. En lo que se refiere a la introducción de datos, debe trabajarse más las interfases con el usuario. Por ejemplo, habrá que integrar mejor las imágenes con el sonido, como es el caso del registro y emisión de voz. También habrá que dedicar atención a la estructura del lenguaje natural y a las materias emparentadas de que trata la inteligencia artificial para llegar a mantener conversaciones con los ordenadores. Deberán diseñarse métodos que faciliten el control táctil en la exploración de la "palpación" de los objetos mostrados en pantalla.



Programación del tratamiento de la información

Sólo si se accede a los datos de manera rápida y comprensible importa almacenar gran volumen de información. La programación que sirva a ese propósito reflejará la estructura de las bases de datos y su almacenamiento

Michael Lesk

Quienquiera que tenga su oficina en desorden sabe que disponer de gran cantidad de información no le garantiza el acceso fácil a un dato determinado. En las dos últimas décadas se ha asistido a un rápido incremento de la capacidad de almacenar información en máquinas electrónicas, acompañado de su correspondiente abaratamiento. En general, el desarrollo de programas que organicen y extraigan información almacenada en medios electrónicos no ha seguido ese mismo ritmo. Quienes se dedican a escribir programas para el tratamiento de datos van siempre a la zaga de las posibilidades de la maquinaria de cálculo.

¿Qué principios guían su esfuerzo? En primer lugar, la mejor forma de organización depende del contenido de la información y de cómo se vaya a utilizar. Por ejemplo, se han escrito muchos programas para almacenar listas de nombres; varían considerablemente según la información que deba ir asociada a cada nombre y la manera en que se recuperen. El sistema comercial Soundex se emplea en la identificación de los pasajeros que tengan reserva para un vuelo determinado. Soundex almacena los nombres fonéticamente, lo que reduce el número de errores de transcripción y permite encontrar un apellido aunque se desconozca su deletreo. El Chemical Abstracts Service mantiene programas para determinar si una sustancia se ha identificado ya, tarea que, en sentido formal, es similar a la realizada por Soundex. Pero en este caso los sistemas no son fonéticos; almacenan una considerable información sobre estructuras químicas y nomenclatura. La información específica sobre una rama del conocimiento puede mejorar la eficacia de un programa en el desempeño de una tarea, pero resta flexibilidad al programa para acometer otros propósitos. Sería una mala elección utilizar programas diseñados para una base de datos química en el listado de las reservas de vuelo.

Otra consideración concierne a la estructura física del medio de almacenamiento. A finales de la década de 1970 se popularizaron los discos magnéticos para almacenamiento de datos. En un disco los datos se graban en subunidades, los bloques, y su acceso es más eficaz si las divisiones lógicas de los datos se aproximan a las de esos bloques. La programación destinada a gestionar gran cantidad de información está condicionada, de este modo, a la estructura de la máquina, por una parte, y al contenido de la información, por otra. Los esfuerzos encaminados a diseñar programas que exploten mejor las posibilidades de las máquinas actuales deben ajustarse, en gran medida, a la necesidad de adaptarse a esas dos condiciones fundamentales.

En un sistema de almacenamiento electrónico de datos se entiende por registro un pequeño grupo de ellos que guarden alguna relación. Por ejemplo, en un fichero que describa las existencias de un supermercado, cada registro incluiría el tipo de producto, la categoría general de mercancía a la que pertenece, el número del pasillo donde se encuentra y el precio. Cada división del registro, así el tipo de producto, se denomina campo. El registro se extrae del fichero electrónico mediante una clave: una etiqueta que coincida con un campo, una parte del mismo o una combinación de campos.

Ciertos campos se emplean como claves más a menudo que otros. Es probable que en la base de datos del supermercado el número de pasillo sirva de clave, pero no el precio; por tanto, el usuario podrá conocer fácilmente los productos expuestos en el pasillo 3, pero no la relación de productos que se venden a 250 pesetas. Otros datos, así el nombre del proveedor del producto, la fecha de caducidad y las existencias, pueden o no usarse de clave. Los programas utilizados para el tratamiento de datos deben facilitar la búsqueda del registro que contenga un valor particular de dicha clave.

Debe señalarse que no tiene por qué tomarse la clave directamente del registro. Al formular un sistema de ayuda al tratamiento de la guía telefónica para AT & T Bell Laboratories, transformé, en sus equivalentes formales, los apodos insertados en los registros. Así, los "Chuck" de los registros se convirtieron en "Charles" en las claves. La transformación, no obstante, era específica del dominio de la guía telefónica: resulta obvio que en una base de datos sobre geografía el monte Billings no debe transformarse en Williamings.

Las distintas relaciones que se establecen entre registros, campos y claves

1. TRAYECTOS MAS CORTOS PARA LLEGAR AL HOTEL PLAZA desde Wall Street, calculados mediante un programa de ordenador que acto seguido dibujó el plano de la parte sur de la isla de Manhattan. Un trayecto, en blanco, minimiza el tiempo del viaje y, el otro, en naranja, minimiza la distancia. Puesto que las calles son de dirección única, cada trayecto empieza con un cambio de dirección. El trayecto que minimiza el tiempo va desde West Street hasta la Décima Avenida, por la que se discurre hasta la parte alta de la ciudad. La información almacenada sobre el tiempo medio en recorrer cada calle le permite al programa especificar el trayecto más rápido. El trayecto que minimiza la distancia empieza en West Street, llegando a la parte alta a través de la Sexta Avenida. Los programas de almacenamiento y procesamiento de la información que atañen al plano los formularon el autor y sus colaboradores. El programa que identifica los trayectos que minimizan el tiempo y la distancia lo escribió Jane Elliott.

nos ayudarán a definir las tres principales formas de organizar los registros electrónicos: jerárquica, en red y de relación. El sistema jerárquico toma su nombre de la ordenación que se establece en los campos del registro. En cada grupo de registros se designa un campo maestro; los demás quedan subordinados a él. Se ordenan consecutivamente los grupos de registros, a semejanza de los peldaños de una escalera, de modo que sólo pueden extraerse datos recorriendo los distintos niveles según el camino definido por la sucesión de campos maestros.

Se han venido empleando bases jerárquicas de datos desde los inicios de la era moderna del cálculo mecanizado, en la década de 1940; se hallarán ejemplos de su utilización en muchos programas de aplicación. Valga de símil la base de datos del supermercado presentada anteriormente. El primer nivel de organización podría incluir una tabla de los números de pasillo y la categoría general de las mercancías ofrecidas en ellos. La categoría de la mercancía sirve de campo maestro. Hay que recorrer la tabla de pasillos y su contenido, y seleccionar una categoría, por ejemplo, verduras, para acceder a las tablas del siguiente nivel.

En el segundo nivel de la jerarquía las tablas indicarían los productos ofrecidos en cada pasillo. Las tablas del tercer nivel informarían del precio de cada producto, su proveedor y la fecha de caducidad. Sólo si se recorre varios niveles de la jerarquía de manera secuencial se alcanza la información asociada a un artículo concreto, su precio, por ejemplo. A menos que se construya otros índices en el fichero, resulta costoso, en términos de recursos de ordenador, pedir información que se desvíe del camino jerárquico, verbigracia, pedir directamente el precio.

El modelo en red es algo más flexible que el jerárquico, pues permite establecer múltiples conexiones entre los ficheros. Gracias a esas conexiones el usuario accede a un fichero sin tener que recorrer toda la jerarquía previa al mismo. Las conexiones auxiliares modifican de modo significativo la estructura vertical de la base de datos. Por ejemplo, en la base de datos del supermercado cabría establecer una conexión entre la relación de pasillos y la lista de precios, de tal modo que pueda conocerse el precio de un producto sin recorrer previamente la tabla intermedia que identifica los productos a la venta en el pasillo.

El modelo de relación (también llamado "modelo relacional"), desarrollado hacia 1970 por E. F. Codd, de la International Business Machines, promete mayor flexibilidad que los demás tipos de bases de datos. Tanto en la organización jerárquica como en la red, ciertas preguntas se contestan más deprisa que otras. Además, la construcción de la base de datos determina qué preguntas resultarán difíciles de contestar y cuáles no. En muchos casos no se dispone de razones suficientes para juzgar a priori cuáles serán las consultas más frecuentes.

En las bases de datos de relación se consigue la flexibilidad suprimiendo la jerarquía entre campos. Todos pueden utilizarse como clave para extraer información. Un registro no es ya un conjunto de entidades discretas de las que una desempeña el papel de campo principal, sino una fila de una tabla de dos dimensiones; cada campo sería una columna. La entrada

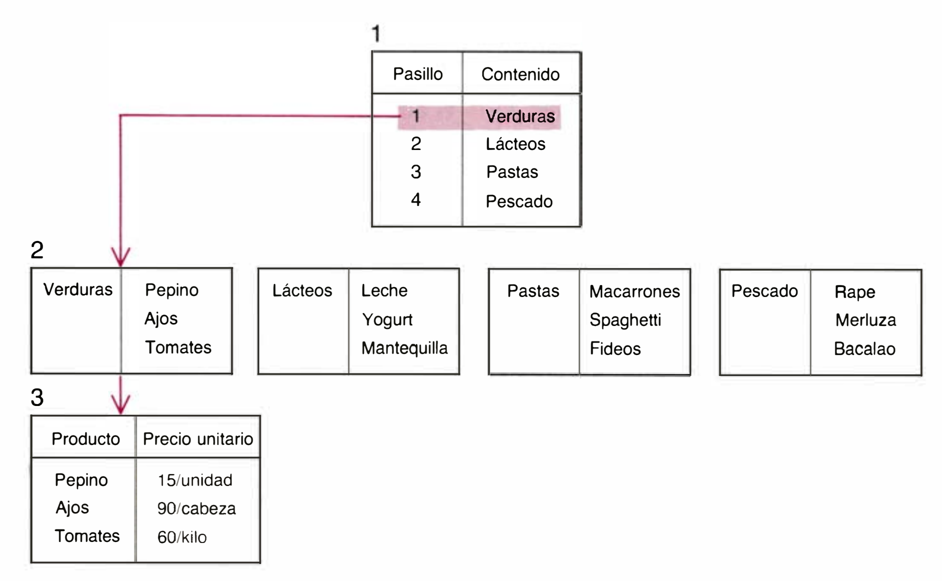
Pasillo 2 Lácteos Leche 70 litro

puede concebirse como una relación entre los cuatro campos siguientes: pasillo, categoría general, producto y precio. La relación podría ampliarse con el nombre del proveedor, la fecha de caducidad del producto, las existencias y otros datos. Del conjunto entero de relaciones se obtienen otras más breves, de sólo dos campos, por ejemplo, seleccionando los campos apropiados; este proceso se conoce por proyección de la relación. Así, se sabe inmediatamente el precio de un producto y nos hacemos con una relación de todos los productos de cualquier pasillo.

En los debates sobre tratamiento de la información suele clasificarse las bases de datos en jerárquicas, en red y de relación. Sin embargo, tal clasificación es menos útil de lo que parece, pues la estructura de cualquier base de datos puede completarse con índices secundarios que facilitan una respuesta eficaz a consultas que no sigan la organización subyacente. Además, ciertos problemas son comunes a los tres tipos de bases de datos. Considérese la tarea de seleccionar un registro de un fichero compuesto por muchos registros relacionados. El fichero podría contener las entradas de un diccionario, las líneas de una guía telefónica u otros registros; en cualquier caso, el problema consiste en extraer un registro lo más rápido posible.

El proceso de identificar un registro aislado depende principalmente del modo en que se organicen los datos en la memoria principal del ordenador o en un medio de almacenamiento secundario, un disco magnético, por ejemplo. Se dispone de técnicas de organización de datos que facilitan posteriores consultas; dichas técnicas pueden aplicarse a cualquiera de los tres tipos de bases de datos.

Si se manejan pequeñas cantidades



2. BASE DE DATOS JERARQUICA. Consta de tablas que deben examinarse en un orden determinado para extraer la información. La figura muestra los pasos necesarios para encontrar el precio de un producto en un fichero que contiene datos sobre las existencias de un supermercado. La primera tabla de la jerarquía identifica los pasillos y la categoría general de las existencias en cada pasillo (*1*). La categoría general se emplea para localizar la tabla del siguiente nivel, que da una relación de los productos que hay en las estanterías (*2*). La tercera tabla contiene el precio de cada producto (*3*). Esta forma de organización sería la adecuada para el dependiente que debe poner precio a cada producto, ya que el método de acceso sigue el recorrido del dependiente por el supermercado. Sin embargo, no sería tan conveniente para responder las preguntas del cliente sobre precios. En la forma de organización denominada base de datos en red, podría establecerse una conexión entre las tablas *1* y *3*. Dicha conexión reduciría el tiempo necesario para encontrar el precio, aunque requeriría más espacio de almacenamiento en memoria.

de datos probablemente no valga la pena mantener una organización determinada. Pueden almacenarse las líneas en un disco de forma arbitraria; para acceder a un dato, un sencillo programa de emparejamiento examinará secuencialmente las líneas hasta la aparición de cierta combinación de símbolos. En el sistema operativo Unix, por ejemplo, se emplea frecuentemente para este propósito el programa *grep*.

Supongamos que se ha almacenado una lista de números de teléfono en un fichero denominado *telnos* y que figuran en ella los de dos compañías suministradoras de gaseosas. En el sistema Unix, el comando *grep* gaseosa provoca la impresión de los nombres y números de ambas compañías. En el fichero *telnos* la palabra "gaseosa" se introdujo antes del nombre de las compañías, aunque el programa *grep* no exige que la clave esté al principio de la línea. El comando *grep bebida < telnos* producirá el mismo resultado si la palabra "bebida" forma parte del nombre de ambas compañías.

Mantener un fichero de términos que no sigan cierto orden ofrece algunas ventajas. Por una parte, ahorra el tiempo y el esfuerzo que habría que dedicar a ordenar los datos al actualizar el fichero. Además, se mantiene considerable flexibilidad, pues no hay que decidir de antemano qué términos se utilizaran de clave al consultar el fichero. Si un usuario decide averiguar todos los números del fichero *telnos* cuyo prefijo sea 957, el comando *grep 957 < telnos* actúa inmediatamente, con independencia de si tal requerimiento se previó o no al crearse el fichero. Si la lista no está ordenada, pueden añadirse nuevos datos al final, sin vernos obligados a reorganizar los ya almacenados. Además, no hay por qué reservar más espacio de memoria que el que ocupan los propios datos del fichero. En sistemas de almacenamiento más complejos, la información necesaria para organizar los datos también ocupa memoria.

El método de almacenamiento desordenado tiene un inconveniente decisivo: la extracción es muy lenta. Cada línea del fichero debe examinarse por separado, lo que significa que la búsqueda de un fichero de 10.000 líneas tarda 100 veces más que la búsqueda en otro de 100. El algoritmo que utiliza el programa *grep* es comparable a buscar un libro en una biblioteca empezando por la estantería más cercana a la entrada e ir examinando los ejemplares uno por uno hasta encontrarlo.

1

Producto	Pasillo	Categoría	Precio	Unidad
Bacalao	4	Pescado	900	kilo
Leche	2	Lácteos	60	litro
Lechuga	1	Verduras	25	unidad
Macarrones	3	Pastas	60	bolsa
Mantequilla	2	Lácteos	80	100 gramos
Pepinos	1	Verduras	15	unidad
Rape	4	Pescado	1900	kilo
Merluza	4	Pescado	800	kilo
Spaghetti	3	Pastas	60	bolsa
Tomates	1	Verduras	60	kilo
Fideos	3	Pastas	60	bolsa
Yogurt	2	Lácteos	25	unidad

2

Pepinos	15	uno

3

Lechuga	Pasillo 1	Verduras	25	unidad
Pepinos	Pasillo 1	Verduras	15	unidad
Tomates	Pasillo 1	Verduras	60	kilo

3. EN LAS BASES DE DATOS DE RELACION, cada entrada consta de una lista de referencias conectadas; puede obtenerse cualquier subconjunto de éstas a partir de la lista completa. En una base de datos de un supermercado, las entradas contendrían el tipo de producto, el pasillo donde se encuentra el producto, la categoría general a la que pertenece y su precio (*1*). Relaciones más específicas, como el precio, pueden derivarse fácilmente del conjunto de todas las relaciones (*2*). También cabe construir una tabla que contenga los productos disponibles en un pasillo y sus respectivos precios (*3*). Las bases de datos de relación resultan interesantes cuando no se conoce de antemano el tipo de consulta más frecuente.

Para acelerar el proceso de búsqueda, el fichero puede distribuirse en algún orden consecutivo, como el alfabético o numérico. Colocar los términos en orden permite sacar una entrada utilizando la técnica denominada búsqueda binaria. La lista se divide en dos mitades y el programa determina en qué mitad se halla el término buscado. Se repite el proceso hasta encontrarlo.

Supóngase que el fichero contiene las entradas de un diccionario y nos interesa consultar la voz "casa". El programa de búsqueda binaria identifica primero el término situado en la mitad del vocabulario, digamos, "legalidad". Al comparar la primera letra de "legalidad" con la de "casa", el programa determina que "casa" se encuentra en la primera mitad del fichero. El punto medio de la primera mitad es "distorsión" y por tanto "casa" se encuentra en la primera cuarta parte del diccionario. Después de la siguiente división, en "boca", la búsqueda sigue por la segunda porción de los términos restantes. Prolongando la secuencia de obtención de puntos medios llegaremos a la voz "casa".

La búsqueda binaria es más rápida que la búsqueda en un conjunto desordenado de datos. Supongamos que el fichero con que se trabaja consta de n términos. La búsqueda en un conjunto desordenado requiere en media $n/2$ operaciones. La rutina binaria emplea, como máximo, alrededor de $\log_2 n$ operaciones para encontrar un término específico (donde $\log_2 n$ es el logaritmo en base 2 de n). Si los datos están ordenados se ofrece otra ventaja: cuando se ha localizado una entrada es muy sencillo averiguar cuáles son las entradas adyacentes, como la palabra que sigue a "casa" en el diccionario.

Sin embargo, la adición de una nueva entrada a un fichero organizado para la búsqueda binaria es un proceso costoso, pues debe mantenerse el orden de aquél. Cualquier nueva entrada se insertará, por término medio, hacia la mitad del fichero, de modo que habrá que mover la mitad de los términos cada vez que se inserte uno nuevo. Otra grave limitación: las entradas deben localizarse únicamente a partir de la clave mediante la cual se haya ordenado el fichero. Cabe duplicar los términos y almacenar varios ficheros ordenados según diferentes claves, pero ello consume mucha memoria.

Otra técnica de extracción de un fichero ordenado se apoya en subgrupos de entradas adyacentes de datos denominados páginas (*buckets*). La primera entrada de cada página se ordena en una tabla que sirve de índice de las divisiones del fichero. Si n términos se dividen entre $\sqrt{n}$ páginas, cada una con $\sqrt{n}$ términos, la exploración lineal del índi-

a

FICHERO DESORDENADO

gaseosa, bebidas excelsior Valencia 242 0412
líneas aéreas nyair 972 221 9300
líneas aéreas usair 622 3201
gaseosa, bebidas carbónicas Martín 356 0273
líneas aéreas aviberia 624 1500
imagen systems (impresoras láser) 496 7200

b

INSTRUCCION: *grep* gaseosa

SALIDA:
gaseosa, bebidas excelsior Valencia 242 0412
gaseosa, bebidas carbónicas Martín 356 0273

c

INSTRUCCION: *grep bebidas* < *telnos*

SALIDA:
gaseosa, bebidas excelsior Valencia 242 0412
gaseosa, bebidas carbónicas Martín 356 0273

d

INSTRUCCION: *grep aéreas* < *telnos*

SALIDA:
líneas aéreas nyair 972 221 9300
líneas aéreas usair 622 3201
líneas aéreas aviberia 624 1500

4. UN FICHERO DESORDENADO consta de entradas almacenadas sin ningún criterio. El cuadro superior muestra uno de ese tipo: una pequeña guía telefónica denominada *telnos* (*a*). Cuando se ejecuta el comando *grep* del sistema operativo Unix, se examina secuencialmente cada línea del fichero, extrayéndose todas las líneas que contengan una clave o combinación de símbolos específica. Mediante la clave *gaseosa* se extraen los nombres de las compañías de la lista que la suministran, clave que se ha introducido al inicio de ambas entradas (*b*). El mismo par de entradas puede extraerse mediante la clave *bebida*, que se encuentra en los nombres de ambas compañías (*c*). Los nombres de las compañías de líneas aéreas de la lista pueden encontrarse a partir de la clave *aérea*, que consta en el nombre de cada compañía citada (*d*).

ce seguida de una exploración de la página apropiada para localizar la entrada requerida necesita poco más de $\sqrt{n}$ operaciones. Aunque $\sqrt{n}$ no es tan bueno como $\log_2 n$, sobre todo si se manejan ficheros grandes, resulta práctico para los pequeños. Además, es cosa sencilla escribir programas para almacenamiento en páginas.

De momento, lo más que se ha logrado en lo relativo a búsquedas en ficheros ha sido limitarlas a $\log_2 n$ operaciones. Cabe mejorar esa cantidad a través del procedimiento denominado de elección arbitraria de elementos (*hashing*). Para comprender las ventajas de ese sistema consideremos la posibilidad de asignar un número a cada entrada del fichero. Si el número puede calcularse rápidamente mediante un sencillo algoritmo cada vez que se desea acceder a una entrada, ésta podrá localizarse directamente en el fichero sin ninguna búsqueda previa.

Con un método sencillo puede crearse una relación numerada que contenga las entradas de un diccionario. A cada letra del alfabeto se le asigna un número, de tal modo que las letras de una palabra designen un único número, que constituya la dirección de memoria para dicha palabra. La contrapartida del método es que el espacio de memoria requerido es inmenso, y casi toda ella permanece vacía; no en vano la mayoría de las posibles combinaciones de letras no forman palabras con sentido. De hecho, las iniciales de las palabras en español no siguen una distribución uniforme. Por ejemplo, son mucho más comunes las que empiezan por *c, s* o *t* que las que empiezan por *q* o *g*. Si asignáramos 100 palabras a 100 direcciones numéricas en función de su primera letra, se acumularían en determinadas entradas, en lugar de ocupar de manera uniforme las entradas del 1 al 100.

La solución más conveniente es formular un algoritmo que asigne un número "pseudoaleatorio" a cada palabra. El deletreo de la palabra determina completamente el número pseudoaleatorio; si bien palabras distintas pueden generar el mismo número. El algoritmo se basa en una expresión matemática denominada función de elección arbitraria. Una posible función de esas podría asignar un valor numérico a cada carácter del alfabeto; el número pseudoaleatorio que determina la dirección se obtendría sumando los valores numéricos de los caracteres que forman la palabra. Si se da con una función de elección arbitraria eficaz, las entradas se distribuyen uniformemente a lo largo de la fila numerada, conocida por tabla de elección arbitraria.

Resulta conveniente dejar vacías cuando menos una cuarta parte de las entradas de la tabla de elección arbitraria. Ello reduce la frecuencia con que el mismo número pseudoaleatorio se asigna a más de una entrada. Cuando se produce tal duplicación (colisión dicen los especialistas), se recurre a un segundo algoritmo para asignar otra dirección a la segunda entrada. El algoritmo podría asignar, por ejemplo, la siguiente dirección de la tabla; otra posibilidad es aplicar una segunda función de elección arbitraria. Si se conoce la distribución de las entradas, lo que no es habitual, puede mantenerse más de las tres cuartas partes de la tabla llenas.

Hay que remarcar que si se decide organizar un conjunto de información en una tabla de elección, la función de elección arbitraria se calcula cada vez que se busca una entrada. El procedimiento es sencillo; por otra parte, tras el cálculo la entrada se extrae sin búsqueda posterior. La elección arbitraria, por tanto, constituye un método muy rápido de extracción. También es sencilla la alteración del fichero, pues las entradas no se almacenan en un orden consecutivo que haya que respetar, moviendo gran cantidad de ellas cada vez que se incorpora una nueva entrada.

La técnica de elección arbitraria simple tiene un inconveniente, que la invalida para muchas aplicaciones. Dada la posibilidad de que se produzcan colisiones, debe conocerse de antemano el número aproximado de entradas para construir una tabla de elección arbitraria del tamaño adecuado. Si se produce una llegada inesperada de gran número de entradas, quizá tengan que recalcularse todas las funciones de elección arbitraria. En la práctica no siempre es posible predecir el tamaño de un conjunto de datos, de ahí que no convenga en esos casos determinar por adelantado el tamaño de la tabla de elección.

Ficheros desordenados, paginación, búsqueda binaria y elección arbitraria constituyen procedimientos ya en uso en la actualidad, particularmente en su aplicación a bases de datos pequeñas, donde la sencillez de programación prima sobre la eficacia. Se han desarrollado dos métodos aplicables a grandes bases de datos: elección arbitraria extensible y árboles-*B*.

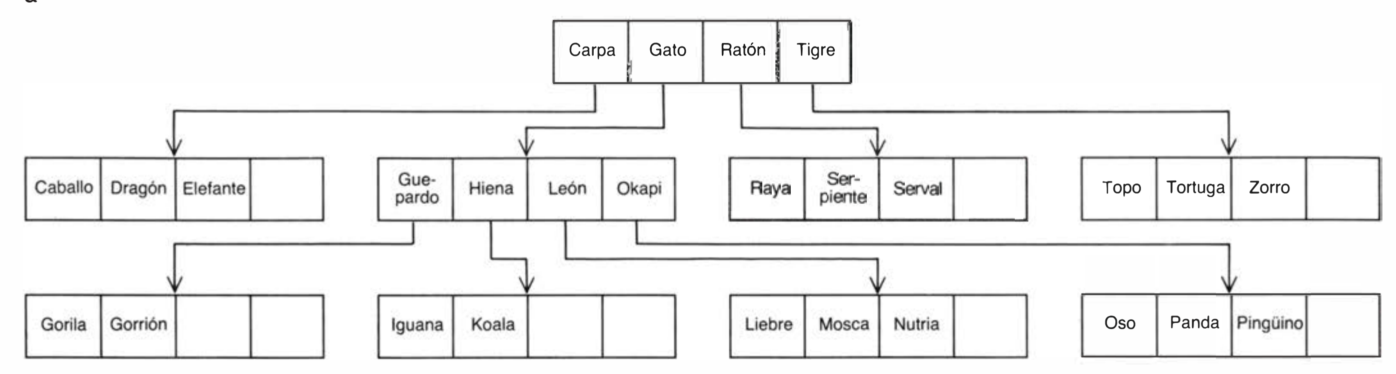
La elección arbitraria extensible se ideó para no tener que especificar por adelantado el tamaño de la tabla de elección. Se calcula un código de elección arbitraria más largo de lo necesario y sólo se utiliza la parte estrictamente precisa para adaptarse al número de entradas en uso; el resto del código se reserva por si el fichero sufre ampliación. Los detalles de la elección arbitraria extensible trascienden el ámbito de este artículo; en síntesis, el objetivo del método es mantener la velocidad de la elección arbitraria junto con un bajo coste de almacenamiento adicional.

En todas las variantes de la elección arbitraria, las entradas del fichero se almacenan en el orden arbitrario determinado por la función de eleccion. Cualquier relación de secuencia que existiera entre los elementos del conjunto de datos original se pierde en el proceso de almacenamiento. Por ejemplo, si se almacena un diccionario en una tabla de elección arbitraria, las palabras que empiezan por *c* se distribuyen al azar por el fichero. Por consiguiente, al extraer una palabra, no resulta fácil averiguar, por ejemplo, cuá-

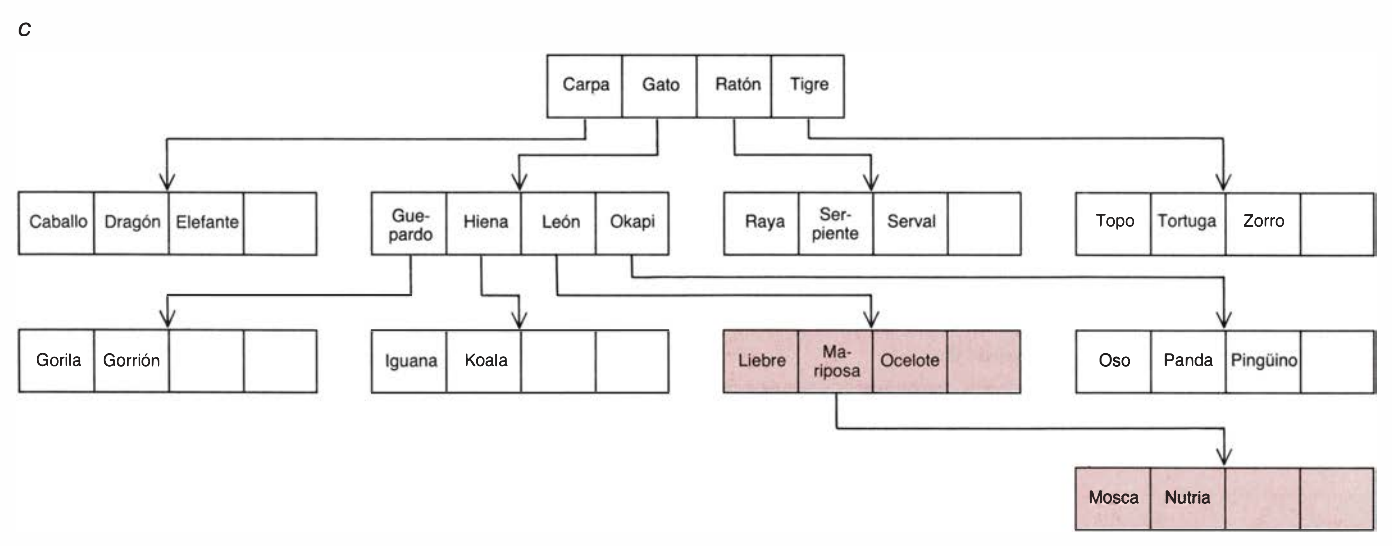
les eran sus adyacentes en la ordenación alfabética.

El árbol-*B* sí permite responder a esa categoría de preguntas. Constituye un mecanismo de aplicación de la búsqueda binaria en el cual las distintas direcciones del fichero se incorporan a la estructura de datos, en lugar de dejarse al cálculo por parte de un algoritmo. El árbol-*B* tiene forma de árbol invertido: muchas categorías (hojas) en el nivel inferior y únicamente una (la raíz) en la cumbre. Cada categoría, o nodo, consta de un conjunto de claves de las entradas de datos.

En el nivel inferior del árbol los nodos contienen un grupo de entradas ordenadas sin omisión. Los nodos del nivel inmediatamente superior contienen la clave de algún nodo del subconjunto de nodos del nivel inferior. Ese proceso de reducción persiste hasta la raíz del árbol, donde sólo queda un nodo. El árbol se recorre de arriba



b
Carpa Gato Ratón Tigre
Caballo Dragón Elefante
Gue-pardo Hiena León Okapi
Raya Ser-piente Serval
Topo Tortuga Zorro
Gorila Gorrión
Iguana Koala
Liebre Mosca Nutria Ocelote
Oso Panda Pingüino



5. ESTRUCTURA DEL ARBOL-*B* y estrategias para añadir entradas al árbol, mostradas para un pequeño fichero formado por los nombres y descripciones de animales almacenados en orden alfabético. El árbol-*B* es una forma de almacenamiento de datos que conserva el orden y que consta de nodos, o pequeños grupos de claves. Cada nodo contiene claves que dividen el fichero, o una parte del mismo, en fracciones (*a*). El nodo cumbre contiene claves que actúan como puntos de división del fichero. Cada clave del nodo cumbre apunta a un nodo del nivel inferior. Las claves de cada nodo de un determinado nivel llenan los huecos que dejan las claves del nivel inmediato superior. Por ejemplo, el hueco que queda entre *gato* y *ratón* se rellena con *guepardo, hiena, león* y *okapi*. La clave *gato* del nodo cumbre apunta al nodo inferior que la incluye. Para encontrar la entrada *liebre* se examina primero el nodo cumbre y se constata que *liebre* queda entre *gato* y *ratón*. Al examinar el segundo nodo, vemos que *liebre* está entre *león* y *okapi*. *León* apunta al nodo que empieza con *liebre*. La clave *liebre* apunta a la entrada que la describe (*no mostrada*). Añadir una entrada a un árbol-*B* puede resultar más o menos sencillo según dónde se encuentren los huecos. Añadir *ocelote* al árbol es inmediato: la entrada se coloca en el hueco vacío que sigue a *nutria* (*b*). Sin embargo, una vez añadido *ocelote*, para agregar *mariposa* se requiere que el nodo que empieza por *liebre* se divida en un nodo superior y otro inferior (*c*). De este modo se conserva la disposición de los punteros: *mosca* y *nutria* rellenan el hueco que queda entre *mariposa* y *ocelote*. (Esquema de Edward Bell.)

abajo utilizando las claves de cada nodo como punteros de nodos del nivel inferior.

Si se almacenara un diccionario en forma de árbol-*B*, el primer nodo contendría, por ejemplo, las palabras "cromosoma", "epicentro", "imposible", etcétera, que constituyen puntos de división del alfabeto. "Cromosoma", la primera clave, podría apuntar a un nodo del segundo nivel que incluyera las palabras "aviso", "binocular", "buzo" y "célebre". Se aprecia fácilmente que esta segunda relación contiene puntos de división desde el inicio del alfabeto hasta la palabra "cromosoma". Cada entrada del segundo grupo apuntará a una relación de resolución más fina. En el nivel inferior del árbol los nodos apuntan directamente a las propias voces del diccionario.

Varias razones han convertido el árbol-*B* en popular. Como se ha visto, permite responder a cuestiones sobre entradas adyacentes cuando ya se ha extraído una. Además, el almacenamiento es relativamente rápido: la búsqueda requiere aproximadamente $\log_2 n$ operaciones, al igual que la adición o borrado de entradas.

Una de las principales razones a las que deben su difusión los árboles-*B* guarda relación con la estructura física del almacenamiento en discos magnéticos. Suele aceptarse como aproximación que cualquier búsqueda en un fichero se toma más o menos el mismo tiempo. Sin embargo, únicamente en la memoria principal las búsquedas tardan exactamente lo mismo, y las bases de datos de interés son lo suficientemente grandes para no caber en la memoria principal. Las grandes bases de datos se almacenan en disco, donde caben dos tipos de acceso con muy distinto tiempo de extracción. El tiempo de acceso al azar es el tiempo medio necesario para extraer un registro a partir de una posición arbitraria del disco. El tiempo de acceso secuencial es el necesario para extraer el registro siguiente al último accedido. En una máquina típica, el tiempo de acceso al azar puede ser 30 veces mayor que el tiempo de acceso secuencial. Por consiguiente, un programa eficaz maximizará el número de accesos secuenciales y minimizará el de accesos al azar, extrayendo grupos relativamente grandes de datos cada vez que se efectúe una búsqueda. Si los datos se almacenan en un árbol-*B*, el tamaño de los nodos de la base del árbol puede ajustarse al tamaño de un bloque del disco. La elección arbitraria no puede sacar partido de la estructura del disco.

En el análisis precedente se han considerado por separado los métodos de extracción y de almacenamiento de datos. Sin embargo, en el diseño de sistemas reales a menudo debe combinarse varias técnicas para alcanzar el mejor resultado operativo; así lo sugieren varios ejemplos extraídos de los trabajos del autor y colaboradores. Hemos ideado un programa experimental que da el pronóstico del tiempo para cualquier ciudad de los Estados Unidos. Si el usuario pregunta por el tiempo de una ciudad determinada, el programa busca el lugar más cercano a la ciudad donde se haya registrado información meteorológica e informa de las observaciones más recientes. A continuación, localiza los pronósticos del tiempo más cercanos y da cuenta de ellos.

Presentar semejante información del tiempo acarrea el uso de tres bases de datos y de programas para cada una de ellas. El sistema se apoya principalmente en un circuito de comunicaciones del Servicio Nacional de Meteorología, que proporciona diariamente seis megabytes de información sobre el tiempo, entre observaciones y pronósticos. Las observaciones se realizan en aeropuertos identificados por un código de tres letras; se necesitan, por tanto, dos bases de datos adicionales a la que contiene la información propia del tiempo. La primera de ellas, una tabla que da la latitud y longitud de cada aeropuerto, se ha tomado de los archivos de la Dirección General de Aviación. La segunda, una tabla que da la latitud y longitud de todas las ciudades del país, la ha proporcionado la Oficina del Censo.

Los datos de las ciudades están almacenados en forma de árbol-*B*. La información del tiempo se guarda en un fichero paginado donde los Estados Unidos se han dividido en pequeños cuadrados definidos por latitud y longitud. Cuando llega un bloque de observaciones del tiempo, el código del aeropuerto se transforma en su ubicación mediante el árbol-*B* y la información se almacena en la página apropiada. Cuando se produce un requerimiento

```
COMMAND:  día

 OUTPUT:  Vier. 4 Mayo 12:55:48 EDT 1984

COMMAND:  tiempo
          elmira, ny

 OUTPUT:  Elmira, NY:(42,093 N, 76,807 O)
          6,3 millas al NO del aeropuerto de Elmira, NY (CHEMUNG COUNTRY) (11:55 AM EDT): tempe-
          ratura 13, humedad 96, cielo cubierto, visibilidad 15 millas

          Próximas 48 horas en Rochester, NY (ROCHESTER-MONROE COUNTRY)
          Hasta las 8 PM EDT/5: máxima 17 mínima 5, prob. precip. hasta 8 AM 30 % hasta 8 PM 10 %.
          Hasta las 8 PM EDT/6: máxima 19 mínima 7, prob. precip. hasta 8 AM 40 % hasta 8 PM 60 %.

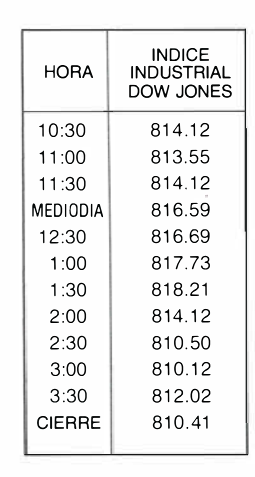
          Pronóstico para el oeste de Nueva York
          Servicio meteorológico nacional buffalo ny
          430 am edt viernes 4 de mayo de 1984
          Lluvia..Intensa en zonas puntuales..Volviéndose intermitente durante el día de oeste a este y
          finalizando esta noche. Máximas durante el día sobre los 13 y mínimas por la noche sobre los 4.
          Nubes y claros durante el sábado. Máximas de 16 a 18.
```

6. INFORMACION METEOROLOGICA para Elmira, Nueva York, según un programa ideado por el autor. Después de comprobar la fecha, el usuario preguntó el estado del tiempo en Elmira. El programa busca el punto más cercano en el que se hayan efectuado observaciones meteorológicas (aquí el aeropuerto de Elmira), y da cuenta de ellas. A continuación va en busca de los pronósticos del tiempo más cercanos. El pronóstico para Rochester se recopiló mecánicamente; el de Buffalo lo prepararon meteorólogos. El programa de información meteorológica, que puede informar del tiempo sobre cualquier ciudad de los Estados Unidos, se basa esencialmente en las observaciones del Servicio Nacional de Meteorología, realizadas en aeropuertos. Se necesitan otras dos bases de datos, además de la que registra la información propia del tiempo: una tabla que da la latitud y longitud de cada aeropuerto y otra que contiene la latitud y longitud de cada ciudad. Cuando se realiza una consulta, el programa identifica la latitud y longitud de la ciudad, descubre los aeropuertos más cercanos e informa de las observaciones y pronósticos.

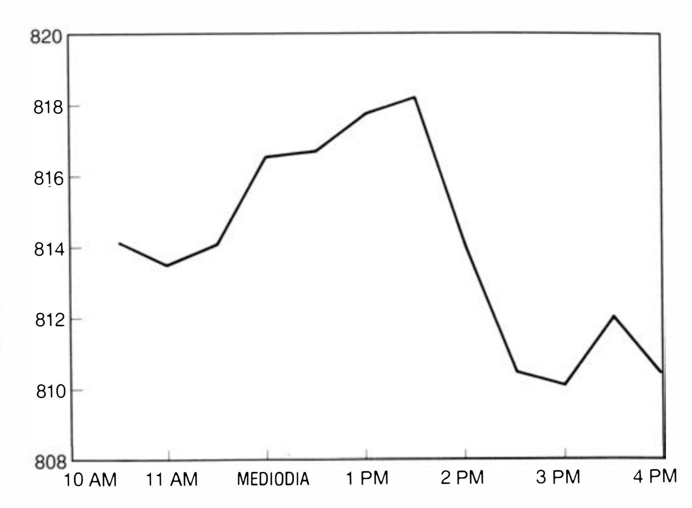
de información del tiempo, se busca la página del aeropuerto más cercano. Si los datos estuvieran en forma unidimensional, y no en dos dimensiones, todo el trabajo podría efectuarse por medio de árboles-*B*; sin embargo, éstos no se adaptan convenientemente a los datos bidimensionales. Este servicio meteorológico lo utilizan frecuentemente mis colegas cuando viajan y desean conocer el tiempo que reina en su localidad de destino.

Hemos montado también un servicio de selección de noticias extraídas de la información que recoge la agencia Associated Press. El programa está aún en experimentación y tiene unos 100 usuarios. Cada día se almacenan unas 200.000 palabras en el sistema informático. Dos son los modos principales de acceso a las noticias. En uno de ellos el usuario escoge noticias actuales a partir de un "menú"; las noticias se seleccionan a partir de unas pocas palabras que sirvan de título y que aparecen en un terminal de ordenador. Por término medio unas 40 personas leen diariamente un total de 840 noticias escogidas del menú.

En el segundo modo de acceso los lectores extraen noticias por medio de un perfil de búsqueda, que se vale de términos específicos, frases u operaciones sintácticas. Quien quiera enterarse del monte Everest puede pedir todas las noticias de la Associated Press en las cuales aparezca la palabra "Everest". El programa también satisface consultas planteadas con varios términos, como "lanzadera espacial", o que se ajustan a ciertas condiciones, como que contengan "teléfono" y "reglamento" en la misma frase. Unas 50 personas utilizan los servicios de consulta y se envían diariamente unas 550 noticias que se adaptan a un perfil determinado.



HORA	INDICE INDUSTRIAL DOW JONES
10:30	814.12
11:00	813.55
11:30	814.12
MEDIODIA	816.59
12:30	816.69
1:00	817.73
1:30	818.21
2:00	814.12
2:30	810.50
3:00	810.12
3:30	812.02
CIERRE	810.41



ORDENADOR	"Las ventas subieron ligeramente durante la mañana de ayer, pero descendieron antes del cierre. Las cotizaciones giraron sobre una actuación mixta, con el mercado anunciando un pequeño descenso de los moderados beneficios."
WALL STREET JOURNAL	"El mercado de valores acabó con resultados mixtos después de su intento de ofensiva en una cuarta sesión inestable de intensas transacciones."

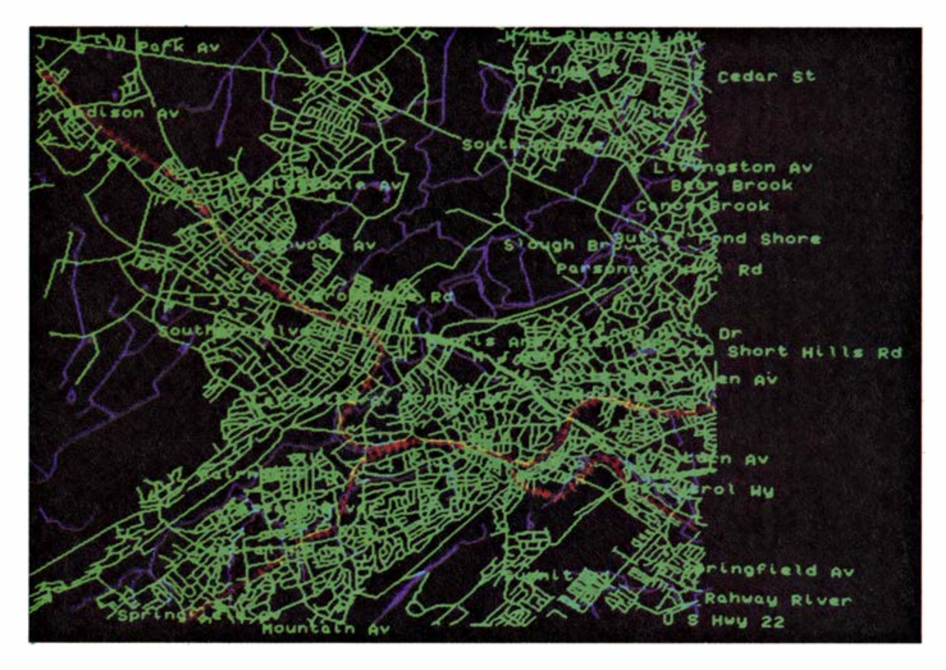
7. SOPORTE LOGICO PARA EL MERCADO DE COTIZACIONES. Comunica los resultados de una jornada de transacciones en números, imágenes visuales o palabras. El índice industrial Dow Jones se ofrece en formato legible por el ordenador; el cuadro superior izquierdo muestra las variaciones del índice el 23 de junio de 1982, en intervalos de media hora. El cuadro superior derecho muestra la salida de un programa que convierte información numérica del índice diario en gráfica. Un programa más complejo creado por Karen Kukich, de la Universidad Carnegie-Mellon, proporciona un informe del índice a lo largo del día. El cuadro inferior compara un informe generado por ordenador con el publicado sobre ese mismo día por *The Wall Street Journal*. El programa no contiene información sobre las transacciones de días anteriores, de ahí que en el informe no quepa incluir una generalización de la tendencia de más de un día de duración. El generador de textos está especializado para una sola tarea específica.

Los sistemas descritos hasta ahora se han diseñado para almacenar y extraer información en forma de números y palabras, pero también cabe procesar grandes cantidades de información derivada de imágenes gráficas. Consideremos el problema de almacenar un plano o guía urbana en un ordenador y luego utilizar dicho fichero para obtener la ayuda que suelen prestarnos los callejeros. La información de los planos se reduce a dos tipos principales de datos: la situación de los nodos y la de las aristas. Un nodo es el punto de intersección de dos calles; una arista es el segmento de calle que conecta dos nodos. La situación de nodos y aristas se convierte en forma digital mediante dispositivos denominados tableros digitalizadores (o scanners), aunque, afortunadamente, en nuestro caso dicha conversión ya la ha realizado la Oficina del Censo. Nuestro sistema de tratamiento de guías se apoya en los datos obtenidos de dicha oficina.

¿Cómo almacenar un plano? Los datos son abundantes y, lo que es aún peor, en lo que atañe al almacenamiento, son bidimensionales. El sistema de almacenamiento debe adaptarse a ambas características. Se dispone de varios métodos capaces de ello; para elegir el adecuado debe precisarse a qué consultas habrá de responder el sistema con mayor eficacia. Cuatro tipos de consulta son los de mayor relevancia: encontrar la situación de un edificio cuando se conocen la calle y el número y el código postal; averiguar si dos calles se cruzan y, si lo hacen, localizar dónde; encontrar todos los puntos directamente alcanzables a partir de un punto dado y, dado un punto y un radio, encontrar todas las calles comprendidas dentro del círculo correspondiente.

Para almacenar datos trazados en dos dimensiones puede acudirse a una matriz de conexiones o a un árbol *k-d*. En la matriz de conexiones la única información que se almacena es la lista de todos los pares de nodos conectados por una arista. Ese fichero no contiene suficiente información para soportar el sistema del plano, pues resulta esencial la inclusión de los nombres de las calles y la situación de los nodos. Deben introducirse los nombres de las calles para que el programa determine cuándo cambia de calle el itinerario. La situación de los nodos debe introducirse para que pueda distinguirse un giro a la izquierda de otro a la derecha y para que pueda dibujarse el plano.

El árbol *k-d* es una variante del *B* que almacena datos en más de una dimensión. En él, descender desde un nivel hasta el inmediato corresponde no sólo a un progresivo refinamiento en la selección de datos, sino también a un cambio de dimensión. En cada paso, los datos se dividen a lo largo de la mayor dimensión. Así, al almacenar un mapa digitalizado de Chile la primera división se haría a lo largo del eje norte-sur, mientras que en un mapa de



Andalucía la primera división recorrería el eje este-oeste.

El árbol *k-d* es un buen medio de almacenar datos bidimensionales, ya que al desmenuzar la información a lo largo de su dimensión mayor se reduce el número de decisiones necesarias para extraer un elemento. Sin embargo, el principal problema en el almacenamiento de mapas o guías no es reducir el número de decisiones a tomar en cada extracción, sino minimizar el número de bloques del disco que debe examinarse. Al igual que la matriz de conexiones, el árbol *k-d* no almacena las calles y sus nombres a la vez, por lo que éstos deben obtenerse de un fichero distinto.

Para acelerar las operaciones de extracción del disco se construyó un "fichero multirregistro" (*patched file*) con dos subficheros, cada uno de ellos con un tipo distinto de información. Un fichero contiene la relación principal de aristas, obtenida de los datos de la Oficina del Censo y de una tabla con información adicional sobre cada calle. Los datos de la Oficina del Censo dan la situación y el nombre de cada arista; la tabla adicional designa las calles de dirección única y los accesos a las autopistas; asimismo, ofrece información sobre límites de velocidad y velocidad media de circulación.

En el fichero principal de segmentos, al igual que en los datos originales de la Oficina del Censo, cada calle se divide en segmentos de longitud tal que sólo se corten en sus extremos, por una parte, y para que puedan representarse por una línea recta, por otra. Un segmento corresponde a un único registro

8. BASE DE DATOS GEOGRAFICA almacenada en forma digital que genera mapas o guías urbanas con varios niveles de detalle. Todas las figuras muestran la misma área: un cuadrado de seis kilómetros de lado centrado en una intersección de Chatham, Nueva Jersey. El plano superior incluye todas las calles del área. El del centro se ha resumido para mostrar únicamente las calles más importantes. El inferior aparece aún más depurado: da por supuesto que las calles siguen una línea recta entre las intersecciones mostradas. El resumen puede reducir considerablemente el tiempo de cálculo necesario para procesar el plano. La representación completa requiere 56 segundos de procesamiento en un VAX 11/750 de la Digital Equipment Corporation (un gran miniordenador). El plano del centro requiere 34 segundos y el inferior 5 segundos. Los programas para procesar planos se apoyan en un fichero que consta de dos subficheros. El fichero principal incluye datos digitales de la Oficina del Censo sobre la situación de nodos y aristas. Un nodo es una intersección y, una arista, la parte de la calle que conecta dos nodos. El segundo es un fichero multirregistro que divide el plano en cuadrados y registra la aparición de nodos y aristas en cada cuadrado. Ambos subficheros se emplean conjuntamente para extraer los mapas y resolver cuestiones sobre itinerarios.

de la base de datos. De acuerdo con esta organización, en la mayoría de las calles cada manzana se guarda en un registro distinto. Los registros se almacenan en orden alfabético por el nombre de la calle.

Cada registro contiene un campo que indica si el segmento es una calle ordinaria, un viaducto de acceso restringido, un acceso a autopista u otro elemento cartográfico, así una vía férrea, un río o una línea limítrofe. Los registros de las calles también guardan los números de los edificios de cada lado del segmento, información sobre la velocidad de circulación, la situación de los extremos y el Código Postal de cada lado del segmento. Mediante una búsqueda binaria en el fichero principal resulta sencillo extraer la situación de cualquier dirección y obtener las intersecciones de cualquier par de calles. Así, el fichero principal responde por sí solo los dos primeros tipos de consulta.

El segundo fichero lo constituye un grupo de segmentos organizados en un formato multirregistro, donde el área cubierta por el plano se divide en cuadrados de 3000 metros de lado. Para dibujar una representación de los segmentos de un área determinada el programa examina secuencialmente la relación de registros; procede luego de igual manera con los registros más pertinentes. El segmento que aparezca únicamente en un registro se almacena una vez, pero el que aparezca en más de uno se almacenará en todos los registros donde tenga un extremo. El fichero multirregistro se deriva en su totalidad de la relación principal. Cuando se realizan cambios, basta con alterar esa lista; el fichero multirregistro se regenera automáticamente.

Seleccionadas las técnicas para organizar y almacenar la información de guías o mapas, debe abordarse la cuestión de la interfase (*interface*) entre el usuario y el sistema. La mayoría de las interfases de ordenador disponibles son de inferior calidad a la de una guía impresa. En muchos mapas de carreteras la relación entre la anchura del menor carácter impreso y la anchura del mapa es aproximadamente de 1 a 1000. Incluso en un terminal de ordenador de gran calidad, esa relación es de 1 a 125. Además, la mayoría de los terminales de ordenador pueden imprimir sólo horizontalmente; unos pocos pueden hacerlo también verticalmente, pero casi ninguno imprime en los ángulos intermedios. Esas restricciones obligan a omitir muchos nombres de calles en las guías elaboradas a partir de información digital.

Para determinar qué rótulos deben omitirse, el programa utiliza información sobre el tamaño y la importancia de las calles. Se da por supuesto que cuanto más larga es una calle, más importante es en lo que atañe al tráfico; suposición bastante acertada en la práctica. También se utiliza información sobre la importancia relativa de las calles para resumir las guías; se representan grandes áreas sin excesivo detalle y sólo se dibujan los itinerarios que recorren calles importantes. Para un cálculo rápido resulta conveniente suponer que cada calle sigue una línea recta entre las intersecciones que sobreviven al resumen.

El sistema que hemos construido responde a los cuatro tipos de consulta mencionados anteriormente. Si se combina la base de datos con un programa escrito por Jane Elliott, de los AT & T Bell Laboratories, puede averiguarse el itinerario más corto entre dos puntos del mapa en términos de tiempo o distancia. La mayoría de los procesadores de mapas utilizan la estructura de fichero multirregistro, análoga al fichero de páginas empleado para datos de una dimensión. El fichero multirregistro es algo más lento que el árbol-*B*, pero explota mejor que éste la estructura del disco. La relación de registros de una única representación suele caber en un bloque de disco, de modo que las extracciones son rápidas. Es más, la estructura multirregistro es de fácil comprensión, uso y actualización.

Son escasas las rutinas convencionales que procesan datos en dos dimensiones; aun así, combinando distintas técnicas se ha resuelto el problema de representación de planos y mapas. Podrían citarse otros interesantes problemas de gestión de bases de datos resultantes de la proliferación de dispositivos que generan información directamente legible por el ordenador. Sin embargo, tal abundancia no ha ido emparejada con el desarrollo de programas de acceso a la información. Sin duda en los próximos años se desarrollarán programas más eficaces e imaginativos para la gestión de información. Aunque los resultados de ese desarrollo no son previsibles, es probable que tal evolución se apoye en los principios establecidos en la introducción de este artículo: la necesidad de adaptar cada programa al contenido de la información, a cómo se va a utilizar y a la necesidad de explotar plenamente la estructura del sistema en que opera el soporte lógico.

MAIN STEAM SYSTEM

ATM ATM
TO FW HTR
TO AUX STEAM PRE HEAT
TO FW PMP TUBES
TO RE-HTR-1
BY PASS COND PUMP
PSI C
STEAM GENERATOR
RCS-
SC-
LVL-
TEMP-
1 A
DEG
PSI
HP TURB
TM1 TH2 -1
TO MSR 1
RPM
TO MSR 2
AD
TURB GLAND
DRN
STEAM GENERATOR
RCS-
SC-
LVL-
TEMP-
2 B
BY-PASS COND DUMP
TO RE-HTR-2
ATM M PSI
RPM
PSI

CEDAR RAPIDS

HAZELTON ### MW
HILLS ### MW
DAEC
VINTON ### MW
MW
HIAWATHA LOAD ### MW
COGGON ### MW
MARION LOAD ### MW
SIXTH STREET LOAD ### MW
BELLE PLAINE ### MW
STONY POINT
SUTLIFF ### MW
MT VERNON ### MW
BLAIRSTOWN ### MW
BERTRAM
PRAIRIE CREEK LOAD ### MW
FAIRFAX LOAD ### MW
HILLS ### MW
N CR E CR
TRIP CLOSE EXEC ACK SWIA WCEN CEN NCEN SCEN INDEX
BERT | 161 KV | <MIN LOW TRIB | XF1 MW | >MAX HI TRIB | OCB 6710 | UNA OPEN

Programación del control de procesos

Esta clase de soporte lógico tiene la función principal de comunicarse con el mundo físico en tiempo real. Un ordenador de control de procesos no fija su propio ritmo, sino que se ajusta a los sucesos que ocurren en el mundo real

Alfred Z. Spector

El número de sistemas de ordenadores que vigilan y controlan procesos en el mundo real está creciendo rápidamente. Estos sistemas se ocupan del control del tráfico aéreo, señalización y seguimiento de los ferrocarriles, funcionamiento de centrales de energía nuclear, distribución del fluido eléctrico, red telefónica, piloto automático y otros sistemas de guía de un avión, control de robots y máquinas-herramienta, funcionamiento de ascensores, condiciones ambientales en el interior de edificios, cadenas de producción en fábricas, vuelo de vehículos espaciales y otros sistemas parecidos. El ordenador de control de procesos se caracteriza, en primer lugar, por su función principal: comunicarse con el mundo físico, en vez de hacerlo con un operador humano (aunque éste puede ser informado sobre el estado del proceso). Otro rasgo distintivo es que este tipo de ordenadores no puede fijar su propio ritmo; debe ajustarse según los sucesos sin limitación que ocurren en el mundo.

Un sistema característico podría ponerse a trabajar controlando una columna fraccionadora que separara componentes químicos ligeros de otros más pesados, como es el caso de una refinería de petróleo. En esta aplicación, el ordenador, dirigido por el soporte lógico, recibe información sobre el nivel y la tasa de flujo de los diversos fluidos, así como sobre las temperaturas y presiones en la columna; emite órdenes para controlar estos factores, determinando así la cantidad y calidad de los productos. El sistema de control podría también programarse para ahorrar el máximo de energía consumida en la planta.

Cualquiera que sea la aplicación, los sensores y accionadores son el enlace entre el ordenador y el proceso. Normalmente, un sensor verifica datos analógicos, cambios en la temperatura por ejemplo, que deben transformarse en datos digitales antes de su presentación al ordenador. Con algunos tipos de sensores el soporte lógico solicita periódicamente información; en otros casos, éste es interrumpido por el sensor para ofrecérsela. Es probable que los programas para controlar procesos incluyan también un dispositivo de medida del tiempo –un reloj– que puede considerarse un sensor. Un accionador manipula eléctrica o electromecánicamente el proceso del mundo real. Controlando la temperatura, un accionador podría conectar o apagar un ventilador.

El enlace entre el ordenador y los operadores humanos son los dispositivos de entrada y salida de la información. El dispositivo estándar de entrada es un teclado. Los sistemas modernos de ordenadores cuentan a menudo con otros métodos de entrada, como un lápiz luminoso o un "ratón", por medio de los cuales el operador puede elegir entre diversas opciones señalándolas en la pantalla. Esta es, por sí misma, un dispositivo de salida, que presenta textos e información gráfica sobre el estado del proceso. Otra forma de salida de la información es una alarma indicadora de que alguna parte del proceso necesita atención.

En el núcleo de un ordenador de control de procesos hallamos un modelo del proceso que ocurre en el mundo real. Modelo que consta de tres componentes: estado patrón, función de actualización del estado y función de predicción. El estado patrón consiste en datos que dan una completa descripción del proceso en cada instante. La función de actualización del estado transforma un estado patrón en otro basándose en la información suministrada por los sensores. La función predicción, si se da un estado patrón correcto, produce un conjunto de órdenes que logran la condición deseada en el proceso que se controla. En términos formales esto se describe como un sistema de control por realimentación: el soporte lógico recibe datos de los sensores, realiza las funciones de actualización del estado y de predicción y emite órdenes a los accionadores. Los resultados de estas órdenes condicionan los datos posteriores que se reciben desde los sensores.

Aparte del modelo, y decisivo para el funcionamiento del sistema, existe un plan estratégico que especifica la secuencia de estados que debe atravesar el proceso bajo control. Por ejemplo, en un sistema de control de tráfico urbano, el plan especifica el estado de los semáforos como función del tiempo y de la fluidez del tráfico. Este plan puede introducirse por medio de operadores humanos o generarse por el soporte lógico a partir de un conjunto de objetivos más abstractos establecidos por los creadores del sistema.

Un dispositivo bastante elemental diseñado para controlar el suministro de calor a un edificio ilustra la estructura de un sistema de control de procesos.

1. SISTEMAS DE ENERGIA ELECTRICA vigilados por soporte lógico de ordenador, que representa en la pantalla el estado del sistema en un momento determinado. En la fotografía superior de la página opuesta, el soporte lógico informa sobre las condiciones en una planta productora con dos generadores de vapor y una turbina. La fotografía inferior muestra el sistema de transmisión de alto voltaje de la Iowa Electric Light and Power Company. Por medio de esta representación, el operador del centro de control de la compañía en Cedar Rapids abre y cierra los interruptores del circuito para redistribuir la energía entre las subestaciones citadas en la representación. El programa fue instalado por Aydin Controls.

Su configuración física incluye un sensor para controlar la temperatura exterior, sensores en varias habitaciones para verificar las temperaturas interiores, un reloj y dos accionadores, que son los interruptores para una bomba de calor y una caldera. Supongamos que el soporte lógico tiene dos objetivos: mantener una temperatura que depende de la hora del día y minimizar el consumo de energía.

El estado patrón contiene la temperatura interior, la exterior y la hora del día. La parte más importante de la función de actualización del estado calcula un promedio sopesado de los datos de los diversos sensores de temperatura interior. La función de predicción utiliza el estado patrón, junto con la información sobre la pérdida de calor del edificio y el rendimiento térmico de los dos calefactores, para predecir cuándo debe apagarse o encenderse cada uno de ellos. La estrategia solicita la determinación de si es más rentable la caldera o la bomba de calor en un momento preciso; esta determinación podría basarse en la temperatura exterior y en el coste del combustible.

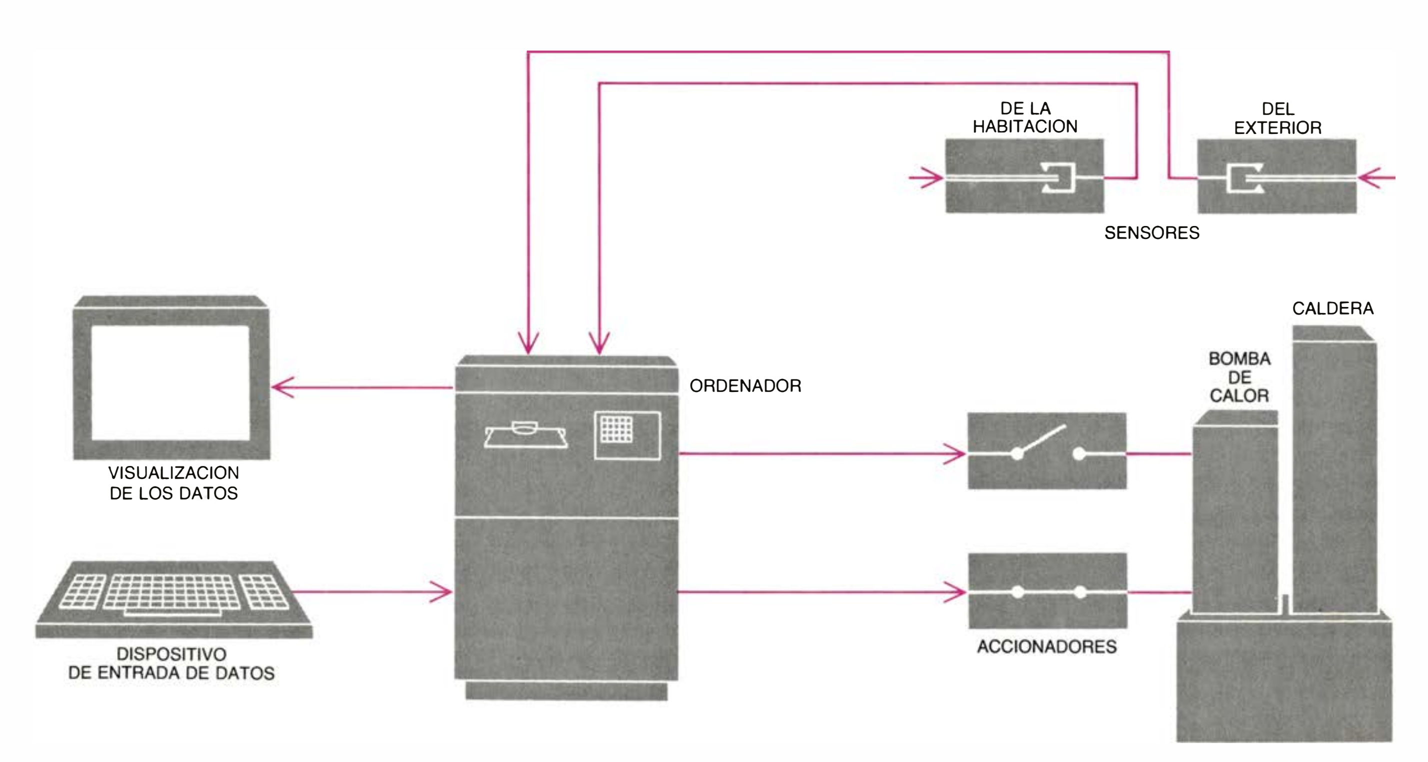
El sistema podría ampliarse añadiendo más sensores (por ejemplo, para verificar los niveles de combustible), teniendo en cuenta otros aspectos del estado patrón y acondicionándolo para que dé cuenta de situaciones anómalas (como un calefactor en mal estado o una ventana abierta). Sin embargo, el carácter fundamental del sistema no variaría con estas mejoras. Seguiría basándose en un modelo del proceso que se controla, empleando una función de predicción para alcanzar nuevos estados.

La mayoría de los sistemas de control de procesos son más complejos de lo que podría sugerir este ejemplo. La principal razón es la propia complejidad del modelo interno. Consideremos un sistema de control de un vehículo, uno elemental en el que sólo se atenga a la aceleración. Para mantener la velocidad correcta, la función de actualización del estado debe hacer una integración numérica para cada lectura de la aceleración. Si se añadiera una cámara para detectar obstáculos y seguir la carretera, se requerirían las más elaboradas técnicas de inteligencia artificial para analizar las vistas con el fin de actualizar el estado patrón.

También pueden ser necesarios procedimientos complejos para llevar a cabo las funciones de predicción y desarrollar los planes estratégicos. La función de predicción que calcula los ángulos de un brazo de robot con seis uniones para colocar y orientar su mano requiere una sustancial cantidad de algebra lineal. Planificar la secuencia de los estados intermedios necesarios para que el brazo del robot se mueva suavemente de una posición a otra es incluso más difícil.

El cálculo y la planificación son tareas que deben realizarse en muchas aplicaciones de los ordenadores; el soporte lógico para control de procesos emplea técnicas que son comunes a otros programas. Pero, por otro lado, los sistemas de control de procesos poseen requisitos propios, diferentes de los exigidos en otras aplicaciones de ordenadores. Uno de esos requisitos está relacionado con la velocidad. Para un ordenador que juega al ajedrez o calcula una nómina, un exacto control del tiempo sólo es crítico en raras ocasiones; una mayor velocidad sería ventajosa, pero un resultado obtenido después de un cierto retraso sigue siendo válido. Sin embargo, un sistema que controla un avión a reacción tiene que tomar decisiones con rapidez; debe actuar en "tiempo real".

El ordenador que juega al ajedrez o que calcula una nómina puede semejantemente realizar una tarea en un momento determinado y programar sus tareas de la forma que le resulte más conveniente. El sistema de control de un avión debe atender múltiples de-



2. SISTEMA DE CONTROL DE PROCESOS para regular el suministro de calor a un edificio. El sistema consiste en dos calefactores (una caldera y una bomba de calor), una serie de sensores para vigilar la temperatura del exterior e interior, accionadores para encender y apagar las unidades de calefacción, un ordenador y un programa con dos objetivos: mantener una temperatura que depende de la hora del día y ahorrar el máximo consumo de energía en todo momento. El programa trabaja según un modelo tripartito de la situación real. El estado patrón incluye las temperaturas interiores y exterior y la hora del día; la función de actualización del estado calcula un promedio sopesado de las distintas temperaturas y revisa adecuadamente el estado patrón; la función de predicción tiene en cuenta elementos tales como el estado del sistema y la tasa de pérdida de calor del edificio para predecir cuándo deben encenderse o apagarse alguno de los calefactores. El programa se basa en la estrategia de usar sólo un calefactor mientras no se necesiten ambos.

mandas a medida que se presentan, por lo que ha de sincronizar cuidadosamente su trabajo en las diversas tareas. La fiabilidad es también más importante en un avión, donde la consecuencia de un error de programación podría ser la pérdida de vidas humanas, no ya la pérdida de una partida o una fuga financiera. La necesidad de velocidad, sincronización y fiabilidad se complica en muchos casos por la organización física del sistema, en la cual ordenadores, sensores y accionadores pueden estar separados espacialmente y funcionando en un medio ambiente estanco.

En la figura 3 un fragmento de un programa sencillo indica los problemas de sincronización y control de tiempo. Es parte de un programa para controlar el sistema de calefacción de un edificio y tiene la forma de un bucle: una secuencia de instrucciones que pueden ejecutarse repetidamente. El único requisito de control de tiempo es que las acciones deben llevarse a cabo con suficiente rapidez para admitir una frecuencia de funcionamiento especificada. Sin embargo, si hubiera que registrar valores de múltiples sensores o gobernar múltiples accionadores, el flujo de control a través del soporte lógico sería mucho más complejo. Además, si el sistema tuviera que manejar interrupciones asincrónicas de los sensores y emitir órdenes en respuesta a tales sucesos, el soporte lógico no podría organizarse como un conjunto de bucles, sino que precisaría una topología algo más compleja.

La programación del control de procesos se configura normalmente como un conjunto de tareas cooperativas, aunque independientes. Por tarea se entiende una secuencia independiente de instrucciones de ordenador que solicita datos segregados, al menos parcialmente, de los datos que se relacionan con otras tareas. En procesadores múltiples pueden llevarse a cabo tareas plurales, ejecutándose una sola en cada uno de ellos. Una disposición más común, sin embargo, emplea un sistema operativo de multitareas para programar la ejecución de muchas en un único ordenador. Se necesita un análisis cuidadoso para asegurar que las tareas reciban servicio y se cumplan así los objetivos globales del sistema.

Mediante una técnica sencilla se programan las tareas ordenadamente con retorno al punto de partida: cada tarea tiene un turno durante el cual se ejecuta hasta completarse. Con esta técnica, una tarea puede encontrarse con serios retrasos si otra más larga está programada delante de ella. Una segunda posibilidad es una técnica con retorno al punto de partida, existiendo un derecho de prioridad. En esta técnica, cada tarea posee únicamente un tiempo corto de ejecución antes de que el procesador se dirija a otra. Si la primera tarea tiene que hacer algún trabajo más dispondrá de otro período de ejecución en su siguiente turno. Una tercera vía es la programación basada en prioridades; aquí las tareas de prioridad más alta poseen períodos de ejecución más largos o más frecuentes. Un último método importante se denomina programación con límite. En este caso se establece un tope para el desarrollo de cada tarea; el sistema se apresta a programar todas las tareas de manera que alcancen sus objetivos.

Repetir secuencialmente *n* **veces por segundo**
Comienzo de secuencia
Muestrear sensores
Actualizar estado patrón
Planificar siguiente estado deseado del proceso
Predecir cómo alcanzar siguiente estado
Emitir órdenes a accionadores
Fin de secuencia

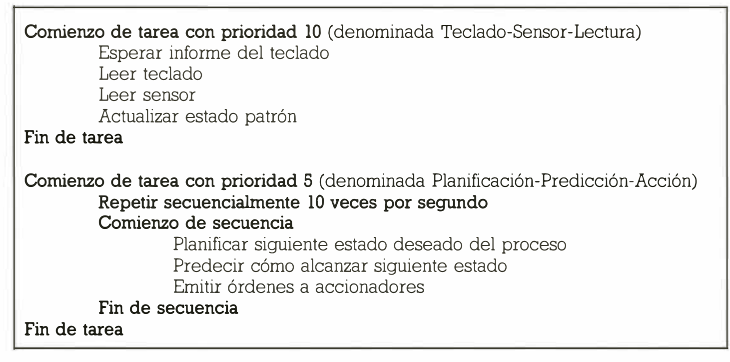
DE LOS SENSORES
PLANIFICACION ESTRATEGICA
OBJETIVOS DEL SISTEMA
A LOS ACCIONADORES
FUNCION DE ACTUALIZACION DEL ESTADO
UTILIZA LOS SENSORES PARA ACTUALIZAR EL ESTADO PATRON
FUNCION DE PREDICCION
EMITE ORDENES A LOS ACCIONADORES
ESTADO PATRON
TIEMPO, TEMPERATURA INTERIOR, TEMPERATURA EXTERIOR Y OTROS DATOS

3. ESTRUCTURA DE SOPORTE LOGICO del esquema de control de procesos para el sistema de calefacción sugerida por el fragmento de programa en la parte superior de esta ilustración. Las órdenes en negrita son palabras clave en el lenguaje de programación; las otras órdenes son advertencias sobre procedimientos del soporte lógico que son específicos del sistema que se controla. El programa tiene la forma de un bucle: es una secuencia de órdenes que puede ejecutarse repetidamente. La parte inferior de la ilustración muestra los componentes principales del programa para controlar el sistema de calefacción.

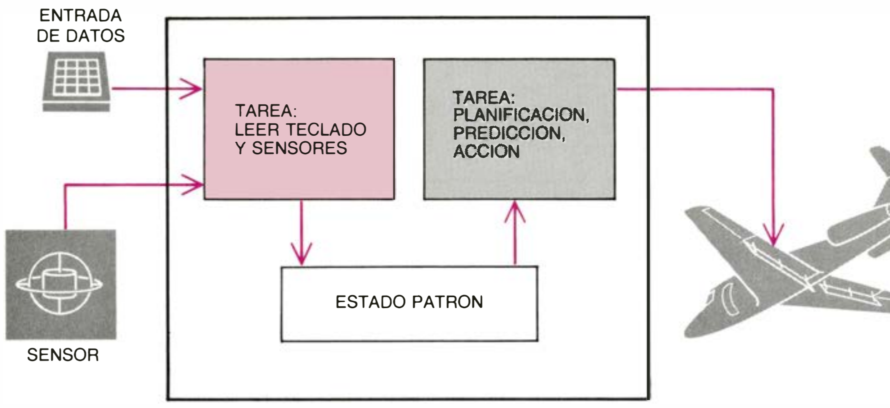
Un fragmento de un sencillo programa de control por multitareas se muestra en la figura 4. Se coordinan dos tareas: Teclado-Sensor-Lectura, con prioridad alta, y Planificación-Predicción-Acción con prioridad más baja. La tarea de prioridad alta recibe los datos de entrada asincrónicamente (es decir, a intervalos impredecibles) desde los sensores y desde un operador humano; la otra tarea ajusta periódicamente los accionadores en respuesta a los datos de entrada más recientes. Si la tarea de prioridad baja está ejecutándose cuando el teclado o un sensor comunica que hay nuevos datos disponibles, el sistema operativo interrumpe la tarea y permite reanudarla sólo después de haber leído los datos. Casi todos los sistemas de control de procesos tienen esta organización de multitareas.

De esta configuración destaquemos que la tarea Teclado-Sensor-Lectura transfiere datos a la de Planificación-Predicción-Acción por medio del estado patrón. La primera tarea simplemente carga información, desde el teclado, en la zona de la memoria donde se almacena el estado patrón, zona a la que también tiene acceso la segunda tarea. Al compartir así la memoria, se logra una potente técnica de comunicación entre tareas, muy eficiente. Este es el procedimiento estándar para las aplicaciones de control de procesos restringidas a un único ordenador.

Por otro lado, la comunicación entre tareas por medio de la memoria compartida complica el procedimiento de multitareas y conduce a algunos problemas interesantes. Por ejemplo, el de la sincronización. Supongamos, en el programa de dos tareas, que la de prioridad baja se interrumpe tras haber leído algunos elementos del estado patrón, y antes de haberlo hecho con otros. Cuando la tarea se reanuda, lee los restantes valores y calcula un resultado que viene determinado por todas


```
Comienzo de tarea con prioridad 10 (denominada Teclado-Sensor-Lectura)
        Esperar informe del teclado
        Leer teclado
        Leer sensor
        Actualizar estado patrón
Fin de tarea

Comienzo de tarea con prioridad 5 (denominada Planificación-Predicción-Acción)
        Repetir secuencialmente 10 veces por segundo
        Comienzo de secuencia
                Planificar siguiente estado deseado del proceso
                Predecir cómo alcanzar siguiente estado
                Emitir órdenes a accionadores
        Fin de secuencia
Fin de tarea
```



4. SISTEMA DE CONTROL DE MULTITAREAS basado en un único programa que atiende y coordina varias funciones; en el ejemplo mostrado aquí guía un avión por control remoto. Un fragmento del programa (*arriba*) incluye dos tareas; una tiene prioridad alta (*color*) y la otra prioridad baja (*gris*). La tarea de prioridad alta recibe entradas de datos a intervalos regulares desde un operador humano, que trabaja con un teclado, y desde los sensores del avión. La tarea de prioridad baja ajusta periódicamente los accionadores que manejan los controles del avión (en este caso una aleta hipersustentadora –flap– de las alas). La tarea de prioridad alta no puede interrumpirse. Si la tarea de prioridad baja está ejecutándose cuando se reciben entradas de datos del teclado o de un sensor, es interrumpida por el programa. Ambas tareas tienen acceso a la representación del estado patrón en la memoria. La compartición de memoria es estándar para la comunicación entre tareas en sistemas dirigidos por un ordenador.

las lecturas. Durante la interrupción, sin embargo, el estado patrón puede haberse visto alterado por la otra tarea, lo que significa que el cálculo se basa en una quimera de datos viejos y nuevos. Supongamos que la tarea interrumpida estaba midiendo una posición en un sistema de coordenadas x-y-z, y que se interrumpió después de haber leído x, mas antes de hacerlo con y y z; su cálculo se basaría en una posición que nunca existió en la realidad.

Normalmente se usan dos sencillas técnicas entre las varias disponibles que hay para sincronizar el acceso a la memoria compartida. Una depende de la designación de segmentos del código del programa en los que no puede interrumpirse una tarea; pudiendo impedirse fácilmente las situaciones favorables para la interrupción por medio del sistema operativo en unión con la configuración física del ordenador. Volviendo de nuevo al ejemplo de dos tareas, el sistema podría especificar que la tarea de Planificación-Predicción-Acción debe estar libre de interrupción mientras está leyendo las coordenadas x, y y z. La otra técnica importante se denomina enclavamiento ("locking"). Antes de que una tarea obtenga acceso a los datos, debe solicitar permiso al sistema operativo para hacerlo. Cuando la tarea ha terminado con los datos, comunica al sistema operativo que éstos quedan disponibles para su utilización por otras tareas. Se dice que la tarea cierra un seguro antes de poder leer datos en la memoria compartida y debe abrirlo cuando ha finalizado. A través de este mecanismo, el sistema operativo asegura que sólo haya una tarea con acceso a los datos en un momento dado.

Aunque la sincronización del acceso a la memoria compartida no revista especial dificultad conceptual, constituye una fuente fecunda de errores en los programas de multitareas. La razón se debe, en parte, a que los errores de sincronización resultan muy difíciles de desenmascarar, pudiendo pasar escondidos, salvo en contadas circunstancias. El funcionamiento defectuoso de un ordenador, que retrasó la primera misión del transbordador espacial, fue un problema de sincronización altamente complejo, cuya probabilidad se había cifrado en una ocasión de las 65 veces que el sistema se pusiera en marcha.

Los problemas relacionados con la sincronización de la memoria compartida pueden afectar de otras formas a la fiabilidad de un sistema. ¿Qué sucede si un error de programación conduce una tarea hasta un bucle sinfín mientras está en un período de ininterrupción? A menos que se tenga gran precaución en el diseño del sistema, otras tareas pueden cancelarse del todo. Un problema que puede surgir con los seguros es que un conjunto de tareas puede formar un ciclo en que cada una esté esperando a otra para hacer saltar un seguro, con el resultado de que no se ejecute ninguna de ellas. Este tipo de bloqueo aparece en sistemas con sólo dos tareas y dos seguros.

En un sistema de memoria compartida se presentan también dificultades que no guardan relación con la sincronización. Al compartir la memoria las tareas no quedan aisladas unas de otras y resulta difícil limitar los efectos de un fallo. Una ejecución defectuosa puede alterar el estado patrón, interrumpiendo con ello el funcionamiento de otras tareas. La compartición de memoria se hace difícil en una red de ordenadores geográficamente dispersos por otra razón, relacionada con la configuración física más que con el soporte lógico: no parece viable construir una memoria que compartan varios ordenadores distantes entre sí.

Por estas y otras razones algunos sistemas de control de procesos se organizan constituyendo un conjunto de tareas que no comparten implícitamente el estado patrón, pero sí intercambian explícitamente información por medio de la transmisión de mensajes. Cada tarea sigue la pista de los elementos del estado patrón que necesita en su propio segmento de memoria no compartido. El sistema operativo aporta los requisitos para el envío y la recepción de mensajes.

Para aplicaciones sencillas de control de procesos, la organización de paso de mensajes es más compleja y generalmente menos eficiente que la memoria compartida. El flujo explícito de información entre las tareas y el mayor aislamiento de éstas ofrecen todavía ciertas ventajas. Una organización de paso de mensajes es la única alternativa viable cuando un proceso se controla por un sistema de varios ordenadores sin memoria compartida trabajando cooperativamente. No obstante, a pesar de las virtudes del paso de mensajes un programa puede verse bloqueado porque

las tareas estén esperando recibir mensajes de otra de ellas.

Los sistemas formados por varios ordenadores que trabajan mancomunados están incrementando su importancia, no sólo para procesos de control, sino también para cálculos científicos, tratamiento de datos e inteligencia artificial. En algunas organizaciones de este tipo, varios procesadores comparten una memoria común; sin embargo, esto sólo es posible si los ordenadores están físicamente próximos. (Se emplean varios microsegundos para transmitir una señal a lo largo de un kilómetro, lo que constituye un retraso intolerable para el intenso tráfico de datos existente entre un procesador central y su memoria principal.)

Cuando los ordenadores están físicamente alejados, el sistema de control se organiza en un conjunto de procesadores con su propia memoria local; los procesadores, interconectados por canales de comunicación, integran una red. Estos sistemas distribuidos, según se les denomina, se apoyan necesariamente en el paso de mensajes. La distancia entre procesadores, sensores y accionadores distingue unos tipos de sistemas distribuidos de otros, en virtud del efecto que la distancia ejerce sobre la anchura de banda y el tiempo de espera. La anchura de banda es una medida del número de bits que pueden transmitirse por segundo; el tiempo de espera es el retraso entre el envío y la recepción de información. Las distancias más cortas contribuyen a una mayor anchura de banda y un menor tiempo de espera, permitiendo así una interacción más estrecha entre las tareas que ejecutan los ordenadores.

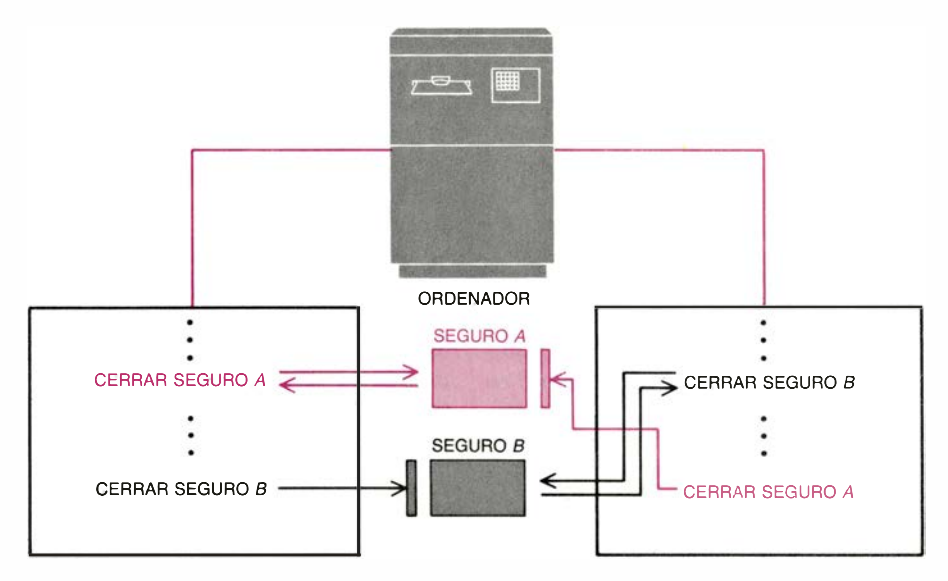
Una razón frecuente que obliga a instalar un sistema distribuido en el control de procesos reside en la propia distribución geográfica de sensores y accionadores. En tal situación, suele ser posible organizar el sistema de suerte que la mayor parte del tratamiento de la información se realice cerca de los sensores y accionadores relevantes; minimizando así la comunicación entre ordenadores. Un ejemplo es un sistema de control de procesos para una fábrica grande con muchos ordenadores semiautónomos en diferentes edificios. Los sistemas intercambian ocasionalmente mensajes con el propósito de coordinar y programar toda la planta, pero suelen trabajar por separado. En un edificio, el sistema podría controlar independientemente la fabricación de un determinado producto y comunicarse con otros sistemas para recoger información sobre los niveles de producción o el suministro de materias primas.

El tratamiento de información distribuido simplifica también el diseño y la instalación del sistema (por ejemplo, reduciendo la cantidad de cableado) y facilita una estructura de organización en la que las personas responsables de un proceso tienen a su cargo las tareas de cálculo asociadas. Una prestación potencialmente mayor es otro motivo importante para los sistemas de multiordenadores. En principio, se pueden realizar más cálculos si varios procesadores trabajan simultáneamente. Un sistema de control de procesos organizado como un conjunto de tareas que se comunican por medio de mensajes es una aplicación natural de este tratamiento paralelo. Se podría ejecutar una tarea de planificación con cálculo intensivo en un procesador de alta velocidad que recibiera datos de tareas que se estuvieran realizando en otros ordenadores y que les devolviera los planes estratégicos.

Quizá la razón más importante para adoptar el cálculo distribuido en un sistema de control de procesos es que se trata de un medio útil de alcanzar fiabilidad. Cuando un sistema está dividido en subsistemas que operan autónomamente, un fallo en una máquina no debería hacer fracasar el sistema entero. La configuración de la fábrica descrita antes sirve de ejemplo: aunque falle un subsistema, los demás pueden continuar, al menos hasta que se les acaben las materias primas.

Si el objetivo es el funcionamiento continuo del sistema entero, el tratamiento de datos redundante es un medio necesario, aunque no suficiente, para tal fin. El sistema completo de control de procesos desde los sensores hasta los accionadores, y no sólo el ordenador y su soporte lógico, debe seguir trabajando a pesar de algún fallo en el funcionamiento. Un ordenador que trabajara correctamente, pero recibiera datos incorrectos de los sensores y emitiera por ende órdenes defectuosas a los accionadores, podría producir más daño que otro que suspendiera su funcionamiento.

En la búsqueda de fiabilidad, un funcionamiento continuo que no se vea en absoluto afectado por fallos constituye el objetivo principal. La naturaleza de los compromisos realizados cuando esa meta no puede alcanzarse está sugerida por el planteamiento de la fiabilidad abordado en los sistemas de control del transbordador espacial. Merced a la redundancia en el sistema principal del ordenador a bordo del transbordador, un fallo aislado no obliga a cambios en la misión. Un segundo fallo no pone en peligro a la tripulación o al vehículo, pero el transbordador torna a la tierra tan pronto como es posible, porque nuevos fallos acarrearían serios riesgos. Así se dice que en el transbordador un

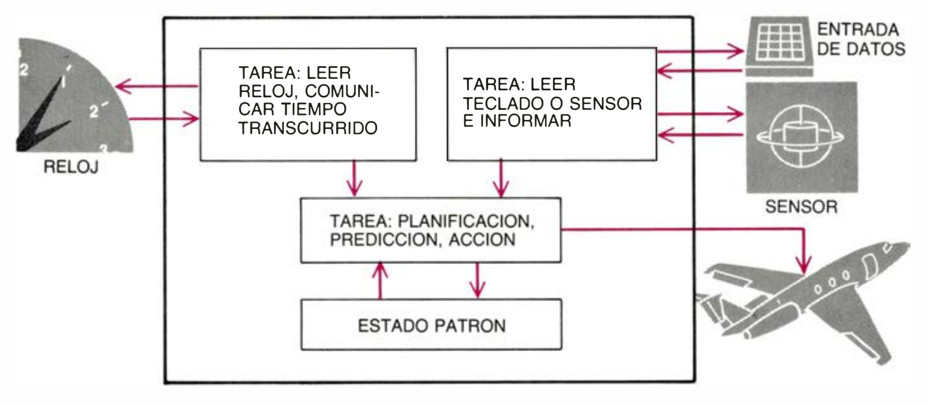


5. PROBLEMA DE BLOQUEO, que puede surgir en un programa que emplee la técnica denominada enclavamiento ("locking") para sincronizar la compartición de la memoria por tareas múltiples. En esta técnica una tarea no puede tener acceso a los datos almacenados en la memoria hasta que cierre un seguro, lo que significa que pide permiso al sistema operativo para acceder a los datos y éste se lo concede. Cuando la tarea ya no necesita el acceso, abre el seguro. Aquí se muestra un bloqueo en un sistema de dos tareas. La de la izquierda está ejecutando una serie de instrucciones (representadas por puntos) cuando encuentra una orden para cerrar el seguro *A* (*en color*). Tras proceder así, sigue con su función ejecutora. Mientras tanto, la segunda tarea ha seguido instrucciones para cerrar el seguro *B*. Ahora, la primera tarea emite una instrucción para cerrar el seguro *B*; no puede hacerlo porque la otra tarea ha cerrado ya ese seguro, por lo que la primera se suspende hasta que el seguro esté libre. Si la segunda tarea emite ahora una orden para cerrar el seguro *A*, el sistema se bloquea y no puede ejecutarse ninguna tarea.

```
Comienzo de tarea con prioridad 10 {Reloj}
        Repetir secuencialmente 10 veces por segundo
        Comienzo de secuencia
                Enviar mensaje (tarea Planificación-Predicción-Acción, Tiempo transcurrido)
        Fin de secuencia
Fin de tarea

Comienzo de tarea con prioridad 10 {denominada Teclado-Sensor-Lectura}
        Esperar informe del teclado
        Leer teclado
        Leer sensor
        Enviar mensaje (tarea de Planificación-Predicción-Acción-Nuevos Datos)
Fin de tarea

Comienzo de tarea con prioridad 5 {denominada Planificación-Predicción-Acción}
        Repetir secuencialmente 10 veces por segundo
        Comienzo de secuencia
                Esperar mensaje
                Si el mensaje es de la tarea de tiempo entonces
                Comienzo de secuencia
                        Predecir cómo alcanzar siguiente estado
                        Emitir órdenes a accionadores
                Fin de secuencia
                        Si no se cumple la condición anterior actualizar estado local con nuevos
                        datos
        Fin de secuencia
Fin de tarea
```



6. PASO DE MENSAJES, una alternativa a la compartición de memoria para sincronizar el trabajo de tareas múltiples en un sistema de control de procesos. Es el único método viable cuando el sistema consta de varios ordenadores separados. Arriba, un fragmento de un programa de paso de mensajes para controlar un avión; el diagrama de la organización del programa está debajo del fragmento del mismo.

fallo de funcionamiento es un fallo con seguridad.

Si el sistema general no puede mantenerse funcionando tras un fallo, es posible que todavía conserve un funcionamiento parcial, propiedad conocida con el nombre un tanto retórico de degradación decorosa. Un sistema que ya no controlara automáticamente un proceso podría todavía aceptar órdenes en el teclado por parte de un operador humano, para que el proceso pudiera controlarse a mano. Si ni siquiera cabe el funcionamiento parcial, el sistema de control debería al menos llevar a cabo una interrupción del proceso en el caso de un fallo importante.

En el soporte lógico pueden presentarse tres tipos de fallos. Primero, que los requisitos del sistema de control estén mal determinados; así, aunque la programación cumple con exactitud sus especificaciones conduce a un funcionamiento erróneo. En la primera misión del transbordador espacial, el conocimiento insuficiente de las características de vuelo del vehículo llevó a una trayectoria de ascenso que podría no haber permitido un aterrizaje de emergencia en España, según establecía el plan de resolución de complicaciones imprevistas. Segundo, es posible que el diseño que fundamenta el soporte lógico o las instrucciones de programación en las que está integrado el diseño no cumpla las especificaciones. El defecto puede ser algo tan simple como un error tipográfico. Tercero, un operador humano puede equivocarse al utilizar el soporte lógico.

La protección frente a posibles fallos es un punto clave en todas las aplicaciones de los ordenadores, pero en el caso de control de procesos la elevada probabilidad de que existan errores en la sincronización y en el control de tiempo, así como el mayor coste potencial de esos errores, hace más importante y difícil la tarea de protegerse frente a los mismos. Pueden emplearse a veces métodos analíticos formales para probar si un programa satisface los requisitos exigidos. En la mayoría de los casos, sin embargo, debe bastar con métodos analíticos menos formales. Un método usual es disponer de expertos en requisitos y en programación que estudian conjuntamente una especificación de requisitos y un programa destinado a cumplirlos.

Al margen de cuán minuciosamente se analice un programa, seguirá siendo necesario comprobarlo. Los métodos analíticos no suelen poseer potencia suficiente para detectar todos los errores imaginables de programación y diseño del soporte lógico. Además, ningún grado de comparación de los programas con los requisitos puede asegurar que éstos sean correctos y suficientes.

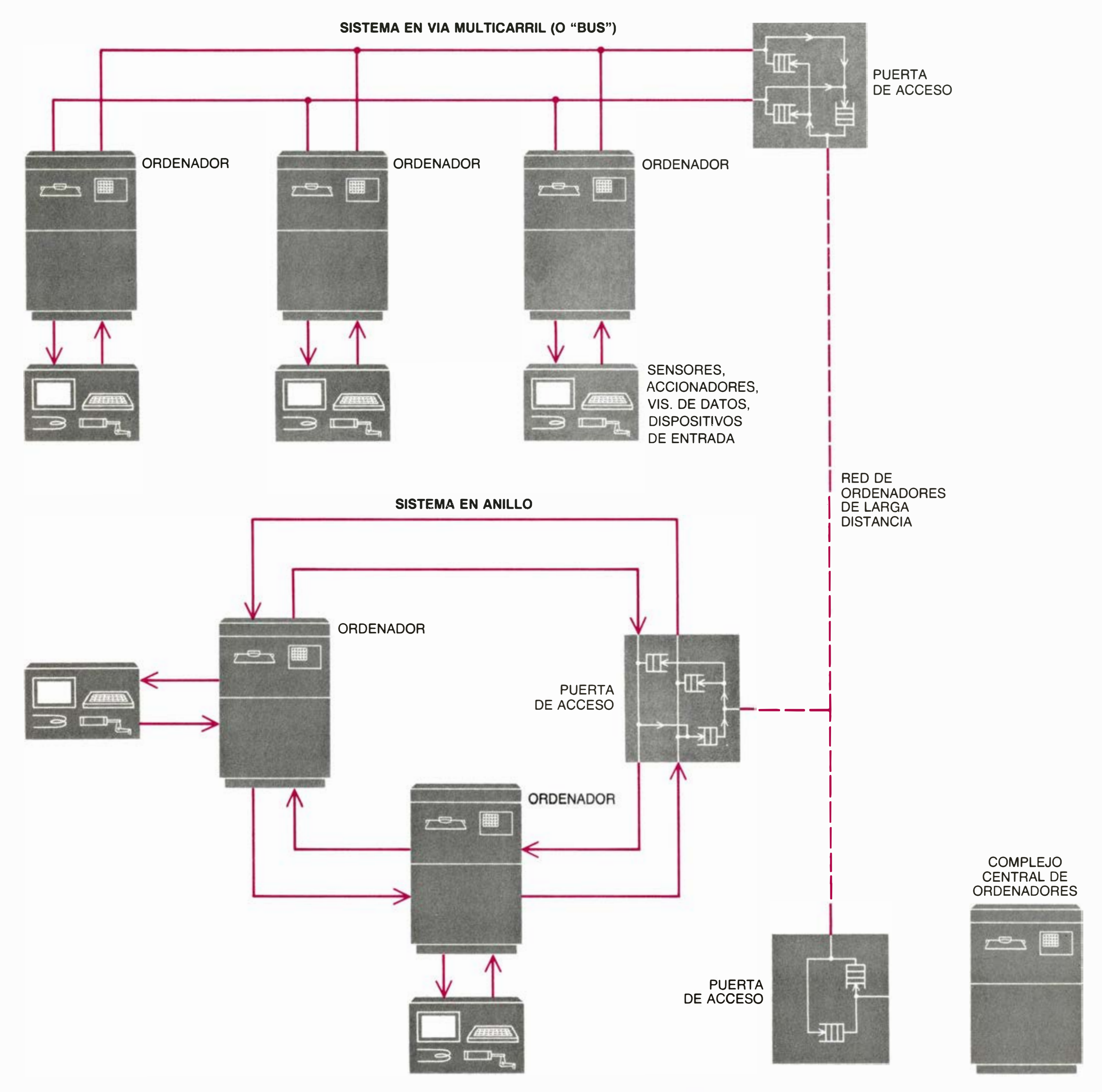
Las personas que verifican los programas se organizan, a veces, en grupo distinto del formado por los programadores; y prueban independientemente el funcionamiento del programa. Durante el desarrollo del sistema operativo principal a bordo del transbordador espacial, el grupo de verificadores de programas, independiente, era aproximadamente del mismo tamaño que el grupo de programación. En un esfuerzo por reducir todavía más el riesgo de errores de programación se forman a veces dos grupos de programación para diseñar independientemente el soporte lógico para la misma tarea. Se presume que al menos una de las dos versiones funcionará.

Para protegerse de los errores de los operadores se necesita un soporte lógico denominado "a prueba de tontos"; la probabilidad de tales errores se reduce al mínimo con un diseño cuidadoso de dicho soporte lógico. La entrada de datos desde el operador puede también comprobarse por razones de verosimilitud; por ejemplo, haciendo determinar al programa si los valores numéricos están dentro del rango adecuado. Otra técnica consiste en dar al operador la oportunidad de reconsiderar las acciones cruciales. El programa puede plantear preguntas como "¿Ha querido usted decir que detenga el proceso?"

Una propiedad interesante del soporte lógico es que, una vez probado que un programa funciona bien, continuará trabajando indefinidamente; el soporte lógico no se "destruye". En contraposición con este hecho, una configuración física que es fiable en un determinado momento puede no serlo posteriormente. Los procesadores pueden pararse o calcular resultados erró-

neos; la memoria y los sensores pueden responder con valores incorrectos; las líneas de comunicación pueden falsear o perder la información y los accionadores pueden dejar de funcionar o volverse imprecisos. Podría fallar un solo componente o inutilizarse el ordenador entero con su memoria y líneas de comunicación, como ocurre en un avión después de un incendio. Aunque el sistema de computación no tenga en sí mismo ningún defecto puede fallar también por efectos ambientales como fluctuaciones en la alimentación eléctrica o calor excesivo. Por tanto, si un sistema ha de ser fiable debería soportar los defectos que surgen aún a pesar de todos los esfuerzos para eliminarlos. La redundancia en diversas formas es el medio de prevenir que tales defectos interfieran en la fiabilidad que se proyecta para el sistema.

Un cierto grado de redundancia puede cimentarse en la configuración física. Muchos ordenadores almacenan información extra con cada dato en la memoria, de forma que algunos errores de almacenamiento puedan corregirse automáticamente. Algunos procesadores vuelven a intentar automáticamente una instrucción que ha fallado, característica de diseño ésta basada en la probabilidad razonable de que el intento tendrá éxito la segunda vez. Los módulos lógicos importantes –incluso procesadores completos– pueden repetirse. Los resultados obtenidos por las unidades que operan en paralelo se comparan; el resultado aceptado se determina



7. SISTEMA DISTRIBUIDO DE ORDENADORES: disposición usual de un sistema de control de procesos cuando los sensores y accionadores están muy diseminados. El sistema mostrado armoniza tres núcleos geográficamente dispersos. Un núcleo está compuesto por tres ordenadores que se comunican entre sí por medio de una red local organizada como una vía multicarril o "bus" (conjunto de conductores paralelos en línea recta). Otro núcleo emplea una red de comunicación en forma de anillo. El tercer componente del sistema es un ordenador central que tiene las funciones de coordinación, compilación de datos y facilitar la intervención de un operador. Los tres sistemas intercambian mensajes a través de una red de comunicación de larga distancia.

por mayoría de votos. La técnica se denomina redundancia modular de n-réplicas. Cuando n es mayor que 3, esta técnica produce un valor correcto si no hay más de $(n-1)/2$ fallos. Mientras los módulos fallen independientemente, la fiabilidad del sistema aumenta al hacerlo n.

Para ganar fiabilidad se emplean también técnicas de soporte lógico. Al igual que la configuración física, el soporte lógico puede volver a intentar una acción después de un fallo. En la mayoría de los sistemas de comunicación, el soporte lógico retransmite datos hasta que recibe una señal de que éstos pasaron. También puede desviarse hacia una fuente redundante, después de que haya fallado una de ellas. En un funcionamiento característico, el soporte lógico detectaría un fallo en un accionador al observar incoherencias en los datos provenientes de un sensor; las órdenes se dirigirían entonces hacia otro accionador.

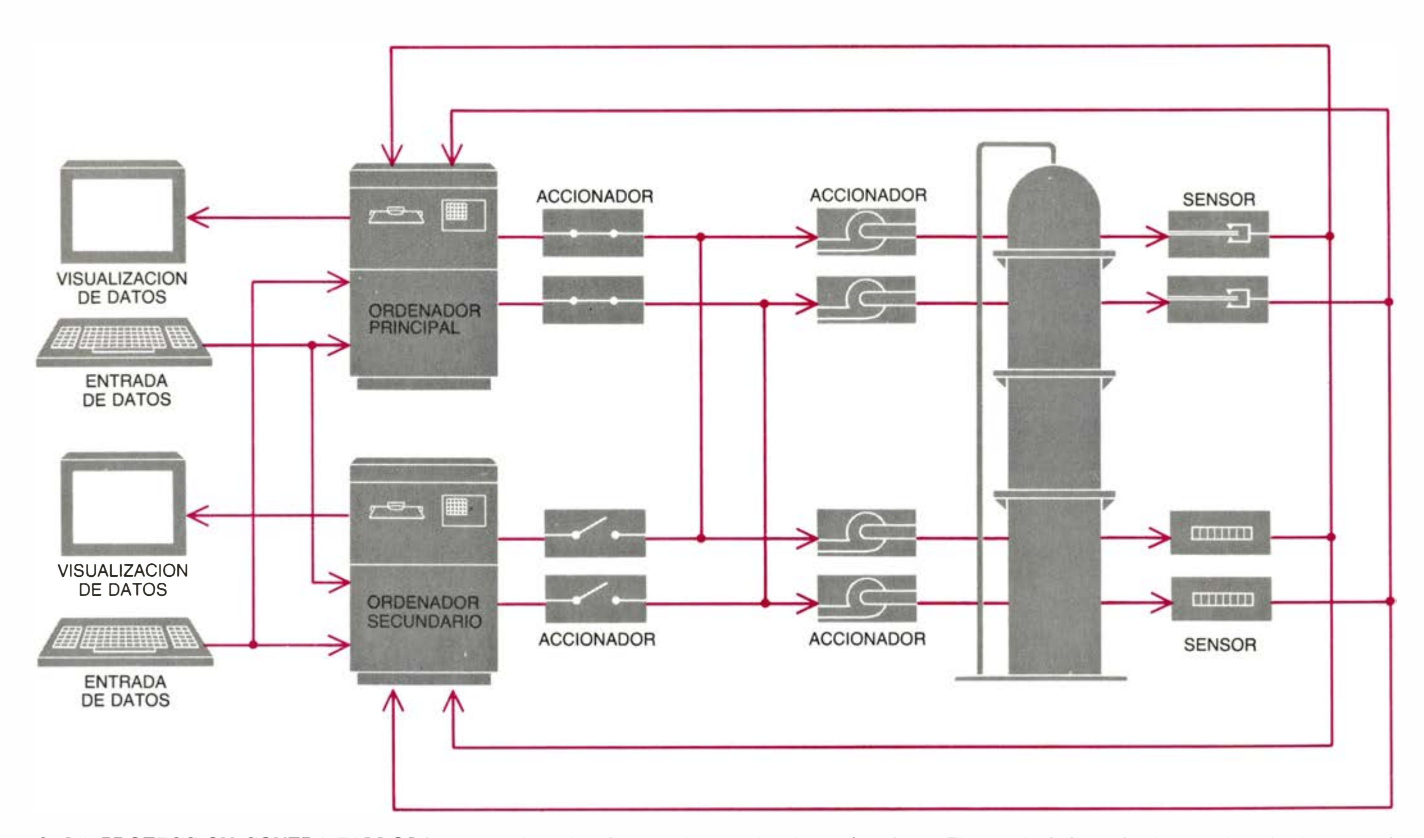
En un sistema de multiordenadores, similares procedimientos de soporte lógico pueden llevarse a cabo en múltiples procesadores para atemperar las consecuencias de un fallo de uno de ellos. El soporte lógico para cada procesador puede programarse independientemente y aumentar así la probabilidad de que al menos una versión del mismo sea correcta. Si el sistema tiene procesadores autónomos que estén alimentados eléctricamente por separado, diseminados, distantes y conectados por canales de comunicación redundantes, hay pocas probabilidades de que pueda fallar todo el sistema. Añadiendo sensores y accionadores redundantes, un sistema de este tipo ofrece una fiabilidad extremadamente elevada.

Una forma de organizar un sistema con redundancia extensiva es designar un ordenador como principal y considerar el resto como secundarios. El principal recibe datos de los sensores y gobierna los accionadores. Los ordenadores secundarios pueden también recibir datos de los sensores, y tendrán, pues, el estado patrón correcto si uno de ellos debe sustituir al principal; pero no gobiernan los accionadores. Los secundarios comprueban el ordenador principal periódicamente, de forma implícita vigilando la coherencia de los datos y de manera explícita mandando mensajes al principal, al que ordenan realizar alguna función de prueba. Principal y secundarios deben filtrar la información de los sensores redundantes; y quizá tener que votar sobre las lecturas de los sensores con el fin de asegurarse de que los cálculos se basan en datos válidos.

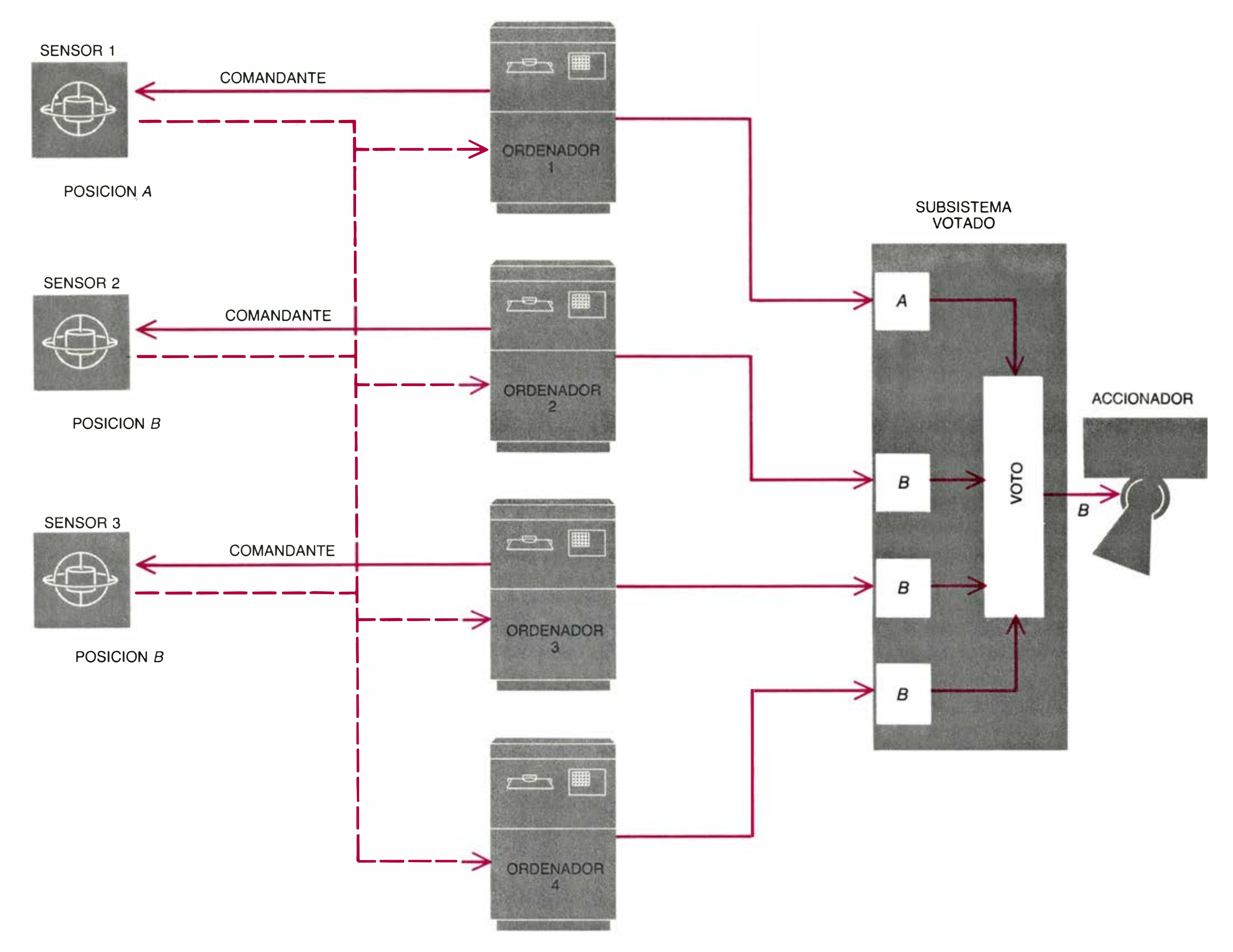
La parte más difícil de esta técnica estriba quizá en determinar verazmente cuándo ha fallado un ordenador principal. Es fácil imaginar una situación en la que un secundario que funcione defectuosamente tomara el control de un principal que lo hiciera correctamente. Se impone establecer un acuerdo entre los múltiples ordenadores secundarios ante la necesidad de que uno de ellos sustituya al principal. Cuando haya tiempo suficiente es más sencillo que un operador humano tome la decisión.

Una forma alternativa de organizar un sistema de multiordenadores se basa en la técnica de votación. Se trata de un procedimiento similar a la redundancia de n módulos; con una salvedad: la votación se hace por módulos de soporte lógico, no por la configuración física. Los módulos realizan cálculos, intercambian resultados entre ellos y votan sobre los mismos. Los procesadores independientes funcionan a velocidades ligeramente diferentes, lo que implica que un grupo de ellos esperará hasta que el último haya terminado un cálculo antes de poder votar.

Un sistema semejante al descrito



8. LA PROTECCION CONTRA FALLOS importa más en los sistemas de control de procesos que en muchas otras aplicaciones de los ordenadores. Se mejora la fiabilidad duplicando los componentes clave del sistema. En este caso, un ordenador se designa como principal y el otro constituye el secundario. Cada ordenador recibe todos los datos, pero sólo el principal está conectado a los accionadores. El secundario hace simplemente los cálculos como si tuviera a su cargo el proceso. Si el principal falla, el control de los accionadores se conmuta al secundario. El sistema también es redundante en sensores, accionadores, visualizaciones de datos y dispositivos de entrada de datos, pues estos elementos deben trabajar con la fiabilidad de los ordenadores.



9. DISPOSICION DE VOTACION presente a menudo en los sistemas de control de procesos para su propia protección frente a posibles fallos. El esquema de votación hidráulica mostrado arriba se emplea en el transbordador espacial para fijar la posición del balancín del cohete. El sistema tiene tres sensores, cuatro ordenadores y cuatro accionadores hidráulicos. Sólo un ordenador emite órdenes para una lectura de cada sensor; pero todos los demás reciben datos procedentes de todos los sensores. Los ordenadores intercambian suficiente información para asegurarse de que están de acuerdo en los datos; si no están de acuerdo, los desechan. Por otro lado, calculan independientemente el valor adecuado de salida por medio de algoritmos idénticos. Cada ordenador gobierna un accionador distinto; si las órdenes entran en conflicto, tres accionadores pueden superar al cuarto; y se cumple, pues, el principio de la mayoría. Si un componente falla, un miembro de la tripulación puede ocuparse del problema, como se indica en el rótulo "Comandante".

Cada uno de ellos gobierna un accionador diferente para cada función, como mover las superficies de los planos de sustentación durante el vuelo en la atmósfera. La votación se hace hidráulicamente: tres accionadores pueden predominar sobre uno solo. Una descripción de accionadores hidráulicos que compiten unos con otros diríase fuera de lugar en un examen de la programación de ordenadores; resulta, por contra, bastante oportuna. La razón del esquema de votación hidráulica débese a que el soporte lógico del control de procesos, por su misma naturaleza, debe influir directamente en procesos que ocurren en la realidad. Control directo que es lo que, sobre todo, distingue los sistemas de control de procesos respecto de otros sistemas de ordenadores, que se limitan a informar sobre resultados de cálculos.

controla elementos de vuelo críticos en el transbordador espacial. El sistema tiene cuatro procesadores autónomos.

Algunos sistemas de control son bastante sencillos. La disposición del control del sistema de calefacción que he descrito constituye un ejemplo de ello. Otros sistemas, como los que controlan el transbordador espacial, los aviones a reacción y los conmutadores telefónicos, están entre los sistemas de ordenadores más complejos jamás construidos. Requieren una avanzada planificación estratégica, alta prestación, elevada fiabilidad y exacto control de tiempo. El cumplimiento de estos objetivos reclama el uso de técnicas en los campos de ingeniería de soporte lógico, arquitectura de la circuitería, diseño de sistemas, sistemas operativos e inteligencia artificial.

Un sistema sencillo puede diseñarse en cuestión de días. Otro de la complejidad del que va instalado a bordo del transbordador espacial necesita años de esfuerzo de miles de trabajadores. Quizás el problema con mayor necesidad de solución sea cómo reducir el tiempo requerido para construir un sistema de control complejo. Casi siempre es posible idear una configuración física y lógica para conseguir un control de procesos, pero esta tarea puede resultar formidablemente cara. Aunque algunas nuevas técnicas de ingeniería de soporte lógico y lenguajes de programación (Ada es uno de ellos) servirán de ayuda, lo que realmente se necesita es conseguir mejores métodos de especificación y programación, medios de análisis y comprobación más sencillos y mejor organización de los sistemas.

Programación en ciencias y en matemáticas

Los ordenadores ofrecen una nueva manera de describir e investigar los sistemas científicos y matemáticos. La simulación por computador puede constituir, en particular, el único camino de predecir la evolución de los sistemas complejos

Stephen Wolfram

Las leyes científicas proporcionan algoritmos o procedimientos para determinar cómo se comportan los sistemas. El programa de ordenador es un medio a través del cual se expresan y aplican los algoritmos. Los objetos físicos y las estructuras matemáticas se representan por números y símbolos en un computador y se escriben programas para manipularlos conforme a los algoritmos. Cuando el programa se ejecuta, modifica los números y los símbolos de la manera especificada en las leyes científicas. Permite, pues, deducir las consecuencias de las leyes.

Ejecutar un programa de ordenador viene a ser como realizar un experimento. Mas, a diferencia de lo que ocurre con los objetos físicos en un experimento convencional, en un experimento por ordenador los objetos no están limitados por las leyes de la naturaleza. Siguen, por contra, las leyes introducidas en el programa, que pueden ser consistentes. De este modo el computador extiende el dominio de las ciencias experimentales: permite realizar experimentos en un universo hipotético. También expande el ámbito de las ciencias teóricas. Las leyes científicas han ido surgiendo por convención en términos de un conjunto particular de funciones y construcciones matemáticas; se desarrollaron a menudo tanto por su simplicidad matemática como por su capacidad de recoger los rasgos relevantes de un fenómeno. Pero una ley científica especificada por un algoritmo puede tomar cualquier forma coherente. El estudio de muchos sistemas complejos, que se resistieron al análisis por métodos matemáticos tradicionales, se hace posible gracias a los experimentos por ordenador y a los modelos que éste permite tratar. El tratamiento por ordenador aparece como una nueva e importante herramienta de las ciencias, complemento de las antiguas metodologías de teoría y experimentación.

Muchos cálculos científicos pueden acometerse todavía por procedimientos tradicionales, sin la ayuda del ordenador. Nadie lo pone en duda. Por ejemplo, dadas las ecuaciones que describen el movimiento de los electrones en un campo magnético arbitrario, cabe deducir una fórmula matemática sencilla que prevea la trayectoria de un electrón en un campo magnético uniforme (un campo con la misma intensidad en todas las posiciones). Sin embargo, para campos magnéticos más complicados no existe tal fórmula matemática sencilla. Pero las ecuaciones del movimiento dan aún un algoritmo por el cual puede determinarse la trayectoria de un electrón. En principio, la trayectoria se podría obtener a mano, pero en la práctica sólo el ordenador puede llevar a cabo el gran número de pasos necesarios para obtener unos resultados precisos.

Para realizar experimentos por ordenador, se puede usar un programa que exprese las leyes del movimiento de un electrón en un campo magnético. Tales experimentos son más flexibles que los tradicionales de los laboratorios. No habría dificultades, por ejemplo, en diseñar un experimento de laboratorio para estudiar la trayectoria de un electrón sometido a la influencia de un campo magnético en un tubo de televisión. Pero ningún experimento de laboratorio podría reproducir las condiciones en que se encuentra un electrón que se mueve en el campo magnético que rodea una estrella de neutrones. El tratamiento por ordenador se puede aplicar en ambos casos.

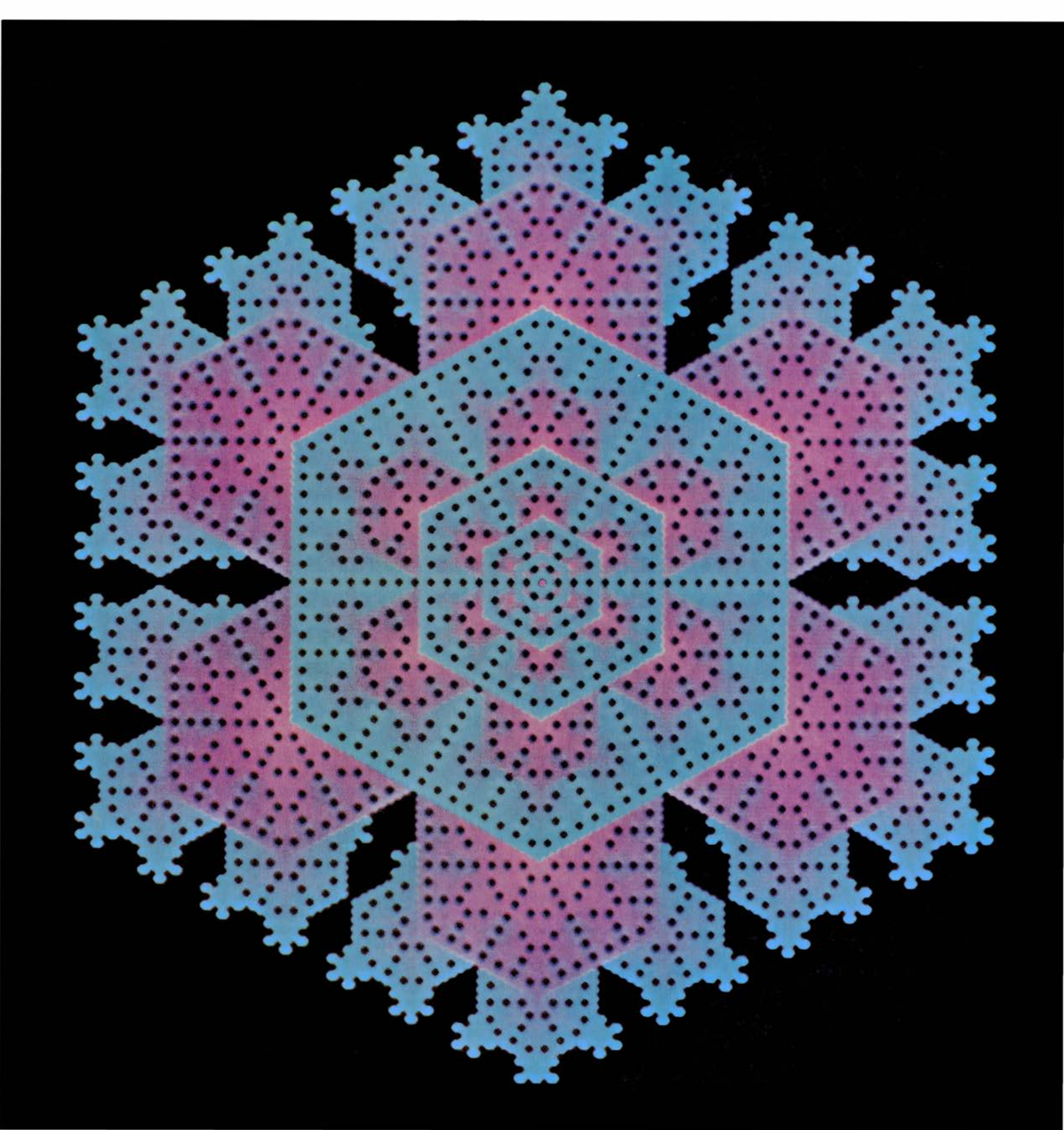
El campo magnético objeto de la investigación se especifica mediante un conjunto de números almacenados en el ordenador. El programa aplica un algoritmo que simula el movimiento del electrón cambiando los números que indican su posición en tiempos sucesivos. Los ordenadores son, ahora, lo suficientemente veloces para que las simulaciones se lleven a cabo con rapidez, lo que permite explorar un gran número de casos. El investigador puede dialogar directamente con la máquina y modificar distintos aspectos del fenómeno a medida que va obteniendo nuevos resultados. El ciclo acostumbrado del método científico, de hipótesis y comprobación de la misma, se recorre ahora mucho más deprisa con la ayuda de un ordenador.

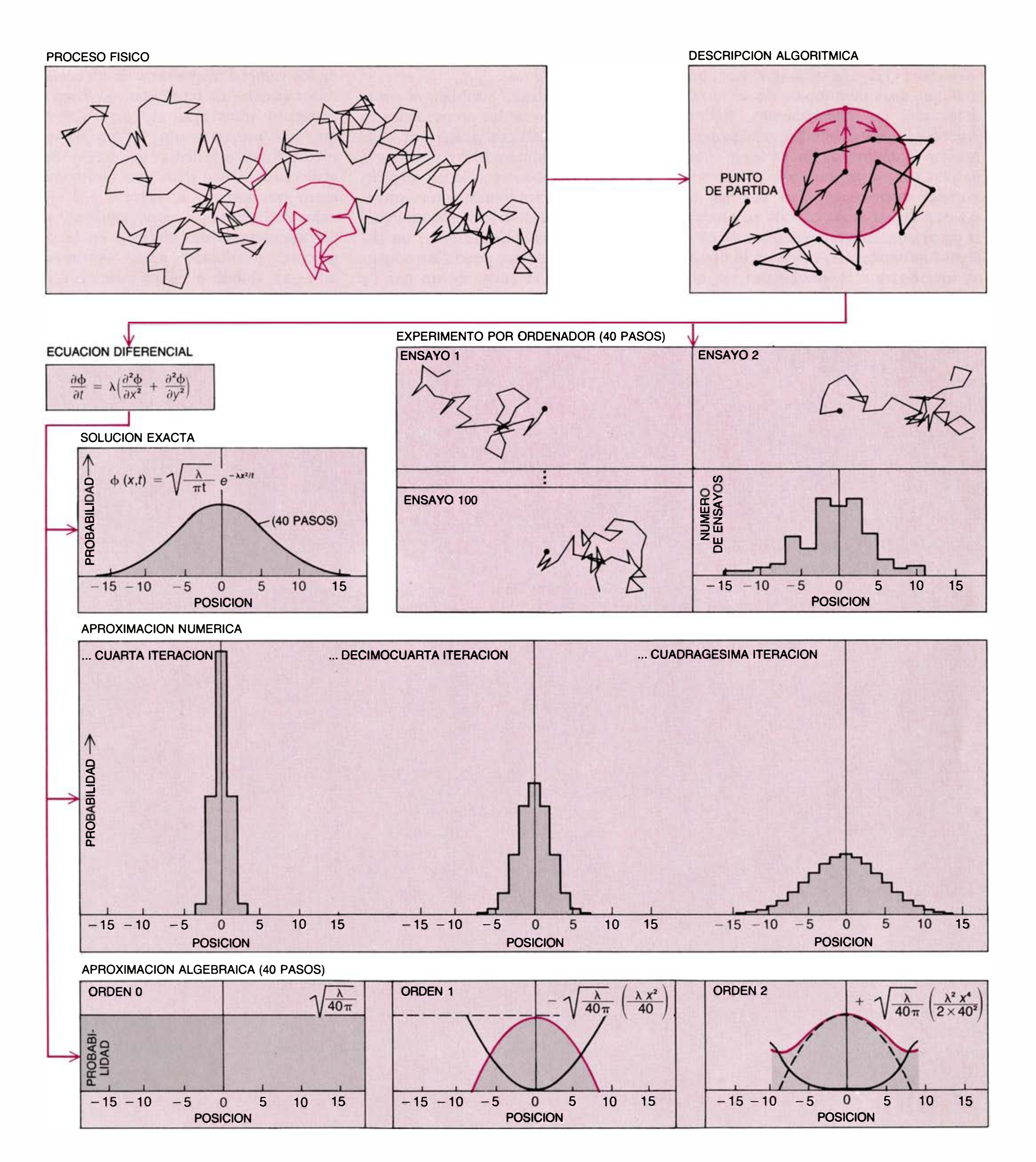
Los experimentos por ordenador no se limitan a los procesos que ocurren en la naturaleza. Un programa puede des-

1. MERCED A LA SIMULACION POR COMPUTADOR se han podido considerar nuevos tipos de modelos para los fenómenos naturales. En la figura, las etapas de la formación de un copo de nieve se generan a través de un programa que traduce el modelo conocido por autómata celular. De acuerdo con el mismo, el plano está dividido en un retículo de pequeñas células regulares hexagonales. Cada célula tiene, o bien el valor 0, que corresponde al vapor de agua (*negro*), o bien el valor 1, que corresponde al hielo (*color*). Partiendo de una única célula roja, en el centro mismo de la ilustración, el copo de nieve simulado se desarrolla a través de una serie de pasos. En cada etapa, el valor resultante de cada célula de la frontera del copo de nieve depende del valor total de las seis células que la rodean. Si el valor total es un número impar, la célula se hiela y toma el valor 1; de no ocurrir tal, la célula se queda en vapor y guarda el valor 0. Se muestran las sucesivas capas de hielo, así formadas, cual secuencia de colores que cambia del rojo al azul cada vez que el número de capas se dobla. Los cálculos requeridos por cada célula son muy sencillos; el modelo de la figura requiere, sin embargo, más de 10.000 cálculos. El único modo factible de generar el modelo es la simulación por ordenador. Esta ilustración se ha realizado con la ayuda de un programa escrito por Norman H. Packard, adscrito al Instituto de Estudios Avanzados de Princeton.

cribir el movimiento de monopolos magnéticos en campos magnéticos, por más que esos monopolos no se hayan detectado en experimentos físicos. Además, se puede modificar el programa con la incorporación de leyes alternativas para el movimiento de los monopolos magnéticos. Una vez más se pueden determinar, cuando se ejecuta el programa, las consecuencias de unas leyes hipotéticas. La máquina le ofrece al investigador la posibilidad de ensayar con un amplio abanico de leyes naturales hipotéticas.

Se puede utilizar, también, el ordenador para estudiar las propiedades de sistemas matemáticos abstractos. Los experimentos matemáticos llevados a cabo en ordenadores pueden sugerir, muchas veces, conjeturas ulteriormente probadas por demostración matemática convencional. Considérese un sistema matemático que pueda introducirse para seguir el curso de un haz de electrones en su movimiento a través de los campos magnéticos de un acelerador circular de partículas. El desplazamiento transversal de un electrón que pasa por un punto, en una de sus revoluciones alrededor del anillo del acelerador, viene dado por cierto número fraccionario x, entre 0 y 1. El valor de la fracción correspondiente al desplazamiento del electrón en la siguiente revolución será, entonces, $ax(1-x)$, donde a es un número que





2. LOS METODOS MATEMATICOS Y COMPUTACIONALES se aplican de formas diversas al estudio de los recorridos aleatorios. Constituyen éstos un modelo para ciertos procesos físicos, así el movimiento browniano de una partícula en suspensión dentro de un líquido. La partícula sufre desviaciones aleatorias al ser bombardeada por las moléculas del líquido; su recorrido puede describirse, pues, como una secuencia de pasos, cada uno realizado en una dirección aleatoria. La manera más simple de deducir las consecuencias del modelo es utilizar el ordenador. Se simulan muchos recorridos aleatorios y se miden sus propiedades medias. El diagrama muestra un histograma donde la altura de cada casilla consigna el número de recorridos aleatorios simulados que han alcanzado una posición de un rango determinado, después de cierto tiempo. Cuantos más ensayos se hacen, la forma del histograma se aproximará tanto más a la de la distribución exacta. Para un recorrido aleatorio ordinario, es posible derivar la distribución exacta directamente. Se puede construir una ecuación diferencial de la distribución, ecuación cuya simplicidad permite el cálculo de la solución exacta. Ahora bien, para la mayoría de las ecuaciones diferenciales, no se puede obtener la solución exacta; hay que recurrir a las aproximaciones: la leve variación de las cantidades en las ecuaciones diferenciales se aproxima con un gran número de pequeños incrementos. Los resultados mostrados en el diagrama se obtuvieron utilizando un programa en que los incrementos espaciales y temporales eran pequeñas fracciones de las longitudes y tiempos de los distintos pasos a lo largo del recorrido aleatorio. Las aproximaciones algebraicas de las ecuaciones diferenciales se calculan como series de términos algebraicos. El diagrama muestra los tres primeros términos de una serie. La contribución de cada término se ilustra en una línea continua de color negro. La línea se añade, por superposición, a la línea negra y punteada que representa la aproximación de orden inferior: el resultado es la aproximación de orden actual (*curva continua en rojo*).

puede variar entre 0 y 4. La fórmula da un algoritmo a partir del cual se puede obtener la secuencia de valores del desplazamiento del electrón.

Bastan unos pocos ensayos para demostrar que las propiedades de la secuencia dependen del valor de a. Si a es igual a 2 y el valor inicial de x se cifra en 0,8, el siguiente valor de x, que viene dado por $ax(1 - x)$, valdrá 0,32. Si se aplica de nuevo la fórmula, se obtiene para x el valor 0,4352. Después de varias iteraciones, la secuencia de valores de x converge hacia 0,5. Realmente, cuando a es pequeño y x cualquier número fraccionario entre 0 y 1, la secuencia se estabiliza rápidamente y da el mismo valor de x para cada revolución del electrón.

Sin embargo, a medida que crece a, se puede observar el fenómeno llamado de doblamiento de período. Cuando a llega a 3, la secuencia empieza a oscilar entre dos valores de x. Mientras a sigue incrementándose, x toma primero cuatro, después ocho valores y, finalmente, cuando a se acerca a 3,57, aparece una serie completa de valores de x. Este comportamiento no podía deducirse fácilmente de la construcción del sistema matemático, pero queda inmediatamente sugerido por el experimento del ordenador. Los pormenores relativos a las propiedades del sistema pueden probarse entonces con una demostración al uso.

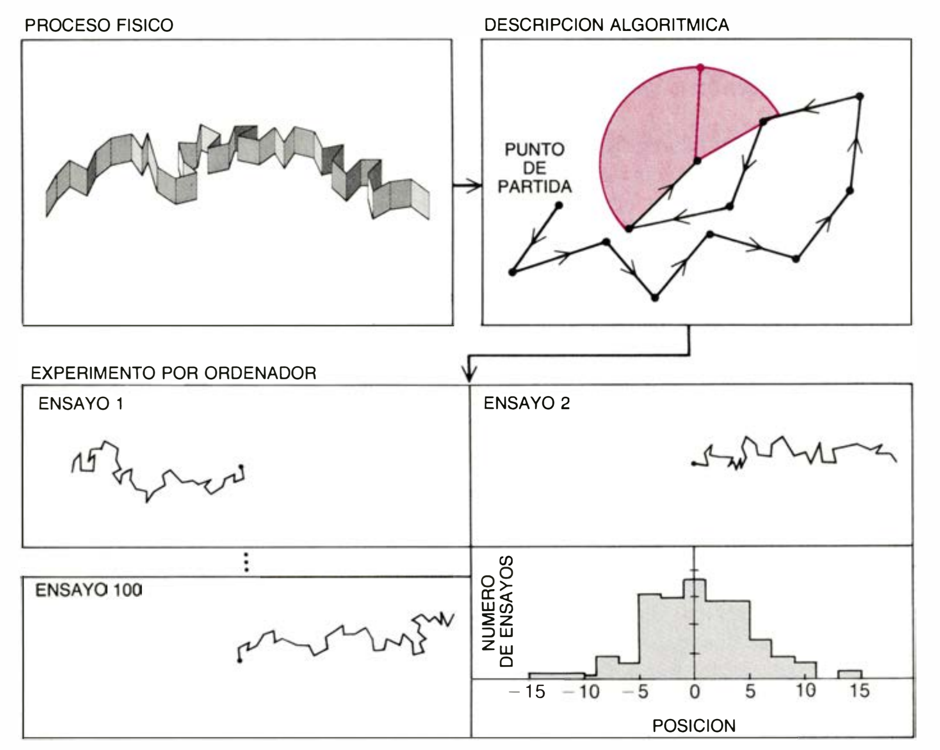
Los procesos matemáticos susceptibles de describirse mediante un programa no se circunscriben a las operaciones y funciones de las matemáticas convencionales. Por ejemplo, no hay ninguna notación matemática tradicional para la función que invierte el orden de los dígitos en un número. No obstante, es posible definir y aplicar la función en un programa. El ordenador viabiliza la introducción de leyes científicas y matemáticas que son intrínsecamente algorítmicas de por sí. Considérese la cadena de acontecimientos que se suceden cuando un electrón, acelerado a alta energía, se dispara contra un bloque de plomo. Hay cierta probabilidad de que el electrón emita un fotón de una energía particular. Y si se emite un fotón, hay, también, cierta probabilidad de que dé origen a un segundo electrón y a un positrón (la antipartícula del electrón). Cada miembro del par puede, a su vez, emitir más fotones, y así termina por engendrarse una cascada de partículas. No hay ninguna fórmula matemática sencilla que pueda describir siquiera los elementos del proceso. Sí puede incorporarse en un programa un algoritmo para el proceso; el resultado del proceso se deducirá ejecutándolo. El algoritmo sirve de ley básica que describe el proceso.

La base matemática de la mayoría de los modelos convencionales de un fenómeno natural es la ecuación diferencial. Tal ecuación establece las relaciones entre ciertas cantidades y sus velocidades de cambio. Por ejemplo, una reacción química se efectúa a una velocidad proporcional a las concentraciones de los reactivos químicos; esa relación se puede expresar con una ecuación diferencial. Una solución a la ecuación daría la concentración de cada reactivo en función del tiempo. En algunos casos simples, es posible encontrar una solución completa a la ecuación en términos de funciones matemáticas convencionales. Sin embargo, en la mayoría de los casos, tal solución exacta no se puede obtener, y se ha de recurrir a la aproximación.

Las aproximaciones más comunes son las numéricas. Supongamos que un término de la ecuación diferencial da la tasa instantánea del cambio de una magnitud con el tiempo. Se puede aproximar el término por el cambio total en la cantidad durante un intervalo pequeño y sustituirlo, entonces, en la ecuación diferencial. La ecuación resultante es, en efecto, un algoritmo que determina el valor aproximado de la magnitud al final de un intervalo, dado su valor al empezar el mismo. Aplicando el algoritmo repetidamente para intervalos sucesivos, se puede hallar la variación aproximada de la magnitud con el tiempo. Intervalos menores proporcionan resultados más precisos. Los cálculos requeridos para cada intervalo son sencillos, si bien en la mayoría de los casos deben repetirse muchas veces para lograr un nivel aceptable de precisión. Tal enfoque sólo se puede realizar con un ordenador.

Los métodos numéricos incorporados a los programas se han empleado para buscar soluciones aproximadas a ecuaciones diferenciales en múltiples disciplinas. En algunos casos, las soluciones tienen una forma sencilla. En muchos otros, muestran un comportamiento complicado, casi aleatorio, aun cuando las ecuaciones diferenciales de donde surgen sean muy simples. Para tales casos, se han de usar matemáticas experimentales.

En aplicaciones prácticas, a menudo se encuentra que no sólo las ecuaciones diferenciales son complicadas, sino que hay también muchas de ellas. Por ejemplo, los modelos teóricos de las explosiones nucleares, empleados en el dise-



3. METODOS COMPUTACIONALES, únicos empleados en el estudio de los recorridos aleatorios sin entrecruzamientos. Los recorridos aleatorios sin entrecruzamientos, que se erigen en modelos para determinados procesos físicos (verbigracia, el enroscamiento de las moléculas de un polímero), difieren de los recorridos aleatorios ordinarios en que cada recorrido debe evitar cruzarse con todos los pasos anteriores. La complejidad impide la construcción de una ecuación diferencial sencilla que describa las propiedades medias del recorrido. No sirven, por tanto, las aproximaciones matemáticas convencionales. Las propiedades del recorrido aleatorio sin entrecruzamiento se deducen de la simulación directa en el ordenador.

ño de armas y en el estudio de las supernovas, involucran centenares de ecuaciones diferenciales que describen las interacciones entre muchos isótopos. En la práctica, tales modelos se usan siempre bajo forma de programas: sólo el ordenador puede seguir las interrelaciones entre tantas magnitudes.

Los resultados de algunos cálculos numéricos, así la abundancia de helio en el universo, se pueden expresar como simples números. En muchos casos, sin embargo, lo que importa es la variación de ciertas cantidades a medida que los parámetros de los cálculos van cambiando. Cuando el número de parámetros es de uno o dos, los resultados se pueden exponer en una gráfica. Si hay más de dos parámetros los resultados se pueden expresar a menudo, y concisamente, a través de una sola fórmula matemática. Lo normal es que no se encuentren fórmulas matemáticas exactas, pero es posible, muchas veces, derivar fórmulas aproximadas. Tales fórmulas revisten particular interés porque, a diferencia de las gráficas y tablas de números, se insertan directamente en otros cálculos.

Una forma habitual de fórmula aproximada es una serie de términos. Cada uno de ellos incluye una variable elevada a determinada potencia; la potencia crece en cada término sucesivo. Cuando el valor de la variable es pequeño, los términos de las series se vuelven cada vez menores; así, para valores pequeños de x, la suma de los primeros términos de una serie infinita, tal como $1-x+x^2-x^3+\dots$, da una aproximación precisa a la suma de la serie entera, que es $1/(1+x)$. Los primeros términos de una serie son habitualmente fáciles de calcular, pero la complejidad de los términos crece muy pronto. Para evaluar términos que incluyen potencias grandes de x el ordenador resulta imprescindible.

En principio, los programas pueden operar con cualquier construcción matemática bien definida. En la práctica, sin embargo, las clases de construcciones que se pueden utilizar en un programa dado están, en gran parte, determinadas por el lenguaje en que se escribe el programa. Los métodos numéricos necesitan sólo un conjunto limitado de construcciones matemáticas; los programas que incorporan tales métodos pueden estar escritos en lenguajes de uso general, verbigracia, C, FORTRAN o BASIC. La deducción y manipulación de fórmulas requiere operaciones sobre estructuras matemáticas de más alto nivel, tal como expresiones algebraicas, para las cuales son necesarios lenguajes de programación nuevos. Entre los lenguajes de este tipo, ahora en uso, está el SMP, desarrollado por el autor.

El SMP es un lenguaje de manipulación de símbolos. Además de operar con números, lo hace también con expresiones simbólicas que pueden representar fórmulas matemáticas. Por ejemplo, en SMP, la expresión algebraica $2x - 3y + 5x - y$ se simplificaría, dando la expresión $7x - 4y$. Esta transformación es general, y se cumple para cualquier valor numérico de x e y. Las operaciones convencionales de álgebra y análisis matemático forman parte de las instrucciones fundamentales del SMP [*véase la figura 5*].

El lenguaje SMP incluye, también, operaciones que permiten definir y manipular estructuras matemáticas de alto nivel, tal como ocurre en el trabajo matemático ordinario. Los números reales (que engloban todos los valores racionales e irracionales), así como los números complejos (que tienen dos partes, la real y la imaginaria) son fundamentales en SMP. Las estructuras matemáticas conocidas como cuaterniones, que son una generalización de los números complejos, no son fundamentales. A pesar de ello, se pueden definir en el SMP, así como especificarse reglas para la suma y multiplicación. En este sentido los conocimientos matemáticos del SMP son susceptibles de crecimiento.

Algunas de las ventajas de un lenguaje del estilo del SMP se pueden comparar con las resultantes de usar una calculadora en vez de una tabla de logaritmos. La difusión por doquier de las calculadoras electrónicas y de los ordenadores hacen obsoletas dichas tablas: conviene mucho más recurrir a un algoritmo de ordenador para obtener un logaritmo que mirar el resultado en una tabla. De forma similar, lenguajes del tipo SMP han hecho posible tener el conocimiento matemático, en toda su extensión, disponible bajo una forma algorítmica. Por ejemplo, el cálculo de integrales, que suele acometerse con la ayuda de un libro de tablas, es tarea dejada ya al ordenador. La máquina no sólo realiza los cálculos finales rápidamente y sin errores, sino que también automatiza el proceso de buscar fórmulas y métodos relevantes.

En el SMP, se ensambla una colección expandible de definiciones que den soporte a una amplia gama de cálculos. Se puede encontrar, en el SMP, la definición de la varianza en estadística, así como la aplicación inmediata de esa definición a un caso particular. Tales defi-

niciones permiten que programas escritos en lenguaje SMP acudan en auxilio de conocimientos matemáticos cada vez más complejos.

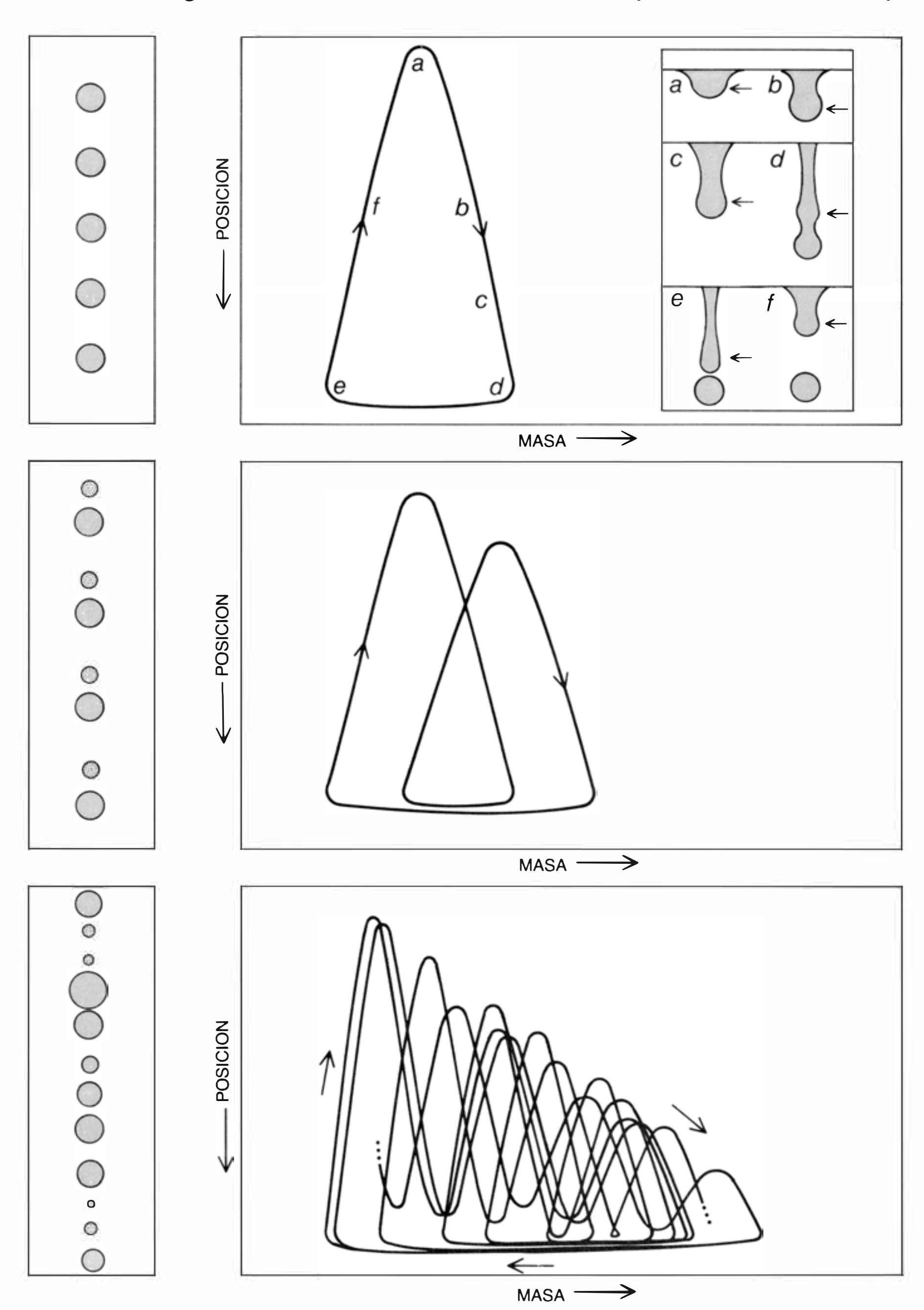
Las ecuaciones diferenciales dan modelos adecuados para todas las propiedades globales de procesos físicos tales como las reacciones químicas. Describen, por ejemplo, los cambios en la concentración total de las moléculas, pero no explican los movimientos de las distintas moléculas. Esos movimientos pueden describirse como recorridos aleatorios: el camino de cada molécula es como el camino que podría tomar una persona en una multitud arremolinándose. En la versión más sencilla del modelo, se supone que la molécula viaja en línea recta hasta colisionar con otra, rebotando en una dirección aleatoria. Todos los pasos en línea recta se suponen de longitud igual. De eso se deduce que, si un gran número de moléculas sigue recorridos aleatorios, la velocidad media con que cambia la concentración de moléculas puede describirse con una ecuación diferencial llamada ecuación de difusión.

Hay muchos procesos físicos, sin embargo, para los cuales no parece factible tal descripción media. En estos casos, no sirven las ecuaciones diferenciales, y se ha de recurrir a la simulación directa. Los movimientos de muchas moléculas o componentes individuales han de seguirse explícitamente; para estimar el comportamiento global del sistema, se averiguan las propiedades medias de los resultados. El único camino viable para llevar a cabo tal simulación es a través del experimento por ordenador; vale decir: ningún análisis de los sistemas que lo requieren podría acometerse sin el ordenador.

El recorrido aleatorio sin entrecruzamiento es un ejemplo de proceso que, según se ve, sólo puede abordarse por simulación directa. Se puede describir por un algoritmo simple, similar al de cualquier paseo aleatorio. Pero difiere en que los pasos sucesivos de un recorrido aleatorio sin entrecruzamientos no deben cortar el camino andado en los pasos anteriores. El enroscamiento de las macromoléculas, así el ADN, son susceptibles de diseñarse como recorridos aleatorios sin entrecruzamientos.

La introducción de esa única limitación complica el recorrido aleatorio sin entrecruzamientos mucho más que el recorrido aleatorio ordinario. La verdad es que no se conoce ninguna descripción media simple, análoga a la ecuación de difusión, para el recorrido aleatorio sin entrecruzamientos. Si se quieren investigar sus propiedades, no parece haber otra opción que llevar a cabo un experimento directo por ordenador. El procedimiento consiste en generar una gran muestra de recorridos aleatorios, eligiendo una dirección aleatoria en cada paso. Se promedian entonces las propiedades de todos los recorridos aleatorios. Este procedimiento es un ejemplo del método de Monte Carlo, así llamado porque su aplicación depende del elemento azar.

Se han presentado varios ejemplos de sistemas cuya construcción es muy



4. COMPORTAMIENTO CAOTICO de muchos sistemas naturales. Recordemos, a modo de ejemplo, el grifo goteando, descrito por un modelo matemático formulado, en función de una ecuación diferencial, por Robert Shaw, del Instituto de Estudios Avanzados. Cuando la velocidad de flujo del agua, a través del grifo, es muy baja, se forman gotas de tamaño constante a intervalos regulares (*izquierda*). El modelo implica que la gráfica de la posición de la cima de cada gota en formación, en relación con la masa de la gota, es una curva simple y cerrada, llamado ciclo límite (*derecha*). La evolución del sistema se representa por un punto que se desplaza sobre la curva en función del tiempo. Si se intensifica el flujo, el comportamiento se vuelve de pronto más complicado. Se produce el fenómeno conocido por doblamiento de período; un par de gotas, a menudo de tamaños diferentes, se forma entonces en cada ciclo. Si el flujo se incrementa todavía más, se produce una secuencia de doblamientos de período adicionales. Por último, justo antes de que el agua que fluye del grifo lo haga de forma continua, se produce una oleada irregular de gotas. Estas alcanzan una gama completa de tamaños; los intervalos entre la formación de gotas consecutivas parecen aleatorios. El comportamiento del sistema se describe, entonces, con una curva irregular llamada atractor extraño, o caótico. La forma de la curva se halla implícita en la ecuación diferencial, pero en la práctica sólo se puede encontrar con las técnicas de aproximación numérica.

ENTRADA	SALIDA	COMENTARIOS
6+17	23	Evaluar una expresión numérica.
6/7+8/9	110/63	Evaluar una expresión numérica con fracciones exactas.
2*x*−3*x*+1	$1-x$	Simplificar una expresión algebraica.
Ex[(*x*−1) (*x*+1)]	$-1+x^2$	Desarrollar expresiones algebraicas compuestas de productos de términos. La notación *x*^*y* representa *x* elevado a *y*. Un espacio entre dos expresiones no numéricas representa una multiplicación.
Ex[(*x*−*a*)^2 (*x*+2*a*)^5]	$8a\ x^6+21a^2\ x^5+10a^3\ x^4-40a^4\ x^3-48a^5\ x^2+16a^6\ x+32a^7+x^7$	
Fac[*x*^2−1]	$(-1+x)\ (1+x)$	Factorizar expresiones algebraicas.
Fac[*x*^6−6*x*^4+4*x*^3+9*x*^2−12*x*+4]	$(-1+x)^4\ (2+x)^2$	
Sol[*x*^2−3*x*+1=0,*x*]	$\{x\rightarrow\frac{3-5^{1/2}}{2},\ x\rightarrow\frac{3+5^{1/2}}{2}\}$	Resolver una ecuación en *x*.
Sol[{*x*+3*a* *y*=4,*y*−15*x*=6*b*},{*x*,*y*}]	$\{x\rightarrow\frac{4}{1+45a}-\frac{18a\ b}{1+45a},\ y\rightarrow\frac{60}{1+45a}+\frac{6b}{1+45a}\}$	Resolver un par de ecuaciones simultáneas en *x* e *y*.
Ps[(1+*x*^3) E^*x*,*x*,0,6]	$1+x+\frac{x^2}{2}+\frac{7x^3}{6}+\frac{25x^4}{24}+\frac{61x^5}{120}+\frac{121x^6}{720}$	Desarrollar en serie la expresión $e^x(1+x)^3$ para *x* próximo a 0, hasta orden x^6.
t:*x*−2*a* *t*^2−2*t*+1	$1+4a-2x+(-2a+x)^2$	Asignar el valor $x-2a$ al símbolo *t*; simplificar la expresión t^2-2t+1 para este valor.
f[2]:6*x*+1 *f*[3]:4−*x* *a* *f*[2]+*b* *f*[3]+*c* *f*[1]	$a\ (1+6x)+b\ (4-x)+c\ f[1]$	Asignar el valor $6x+1$ a *f*[2] y el valor $4-x$ a *f*[3]; evaluar una expresión con *f*[1], *f*[2] y *f*[3], en que *f*[1] no esté especificado.
f	{[2]:1+6*x*,[3]:4−*x*}	Imprimir el objeto *f*, el cual es una lista cuyos elementos están indexados por los números entre paréntesis.
f[1]:7 *f*	{7,1+6*x*,4−*x*}	Asignar el valor 7 a *f*[1]; imprimir el objeto *f*, el cual está dado ahora como un vector o lista ordenada de elementos.
f^2−8	$\{41,-8+(1+6x)^2,-8+(4-x)^2\}$	Calcular el cuadrado; substraer 8 de cada elemento del vector *f*; resulta un nuevo vector.
f[p]:5*x* *f*[*p*^2]:6*x* *f*	$\{[p^2]{:}6x,[p]{:}5x,[1]{:}7,[2]{:}1+6x,[3]{:}4-x\}$	Asignar valores a los elementos de *f* que tienen índices no numéricos; imprimir el objeto *f*.
f[$*x*]:$*x*^2 *f*	$\{[p^2]{:}6x,[p]{:}5x,[1]{:}7,[2]{:}1+6x,[3]{:}4-x,[\$x]{:}\$x^2\}$	Asignar un valor a *f*[$*x*], donde $*x* es una expresión arbitraria; definición general, al final de la lista *f*; se utiliza sólo cuando ningún caso anterior especial se aplica. Imprimir el objeto *f*.
f[*p*]+*f*[2]+*f*[*a*]	$1+11x+a^2$	Evaluar la expresión *f*[*p*] + *f*[2] + *f*[*a*]; la definición general para *f*[$*x*] se aplica para evaluar *f*[*a*].
g[$*x*_=Natp[$*x*]]:$*x* *g*[$*x*−1] *g*[1]:1 *g*	{[1]:1,[$*x*_=Natp[$*x*]]:$*x* *g*[$*x*−1]}	Definir la función factorial *g*[*x*] para los números naturales, siendo *g*[*N*] igual a 1 × 2 × ... × *N*. La definición viene dada por una fórmula recursiva en que *g*[*x*] se especifica en términos de *g*[$*x* − 1]. La expresión $*x*_ = Natp[$*x*] indica que $*x* debe ser un número natural.
g[5]	120	Evaluar *g*[5], factorial de 5.
Abs[3] Abs[−3] Abs[−*x*]	3 3 Abs[*x*]	Calcular el valor absoluto de − 3, 3 y − *x*.
Abs[$*x* $$*x*]:Abs[$*x*] Abs[$$*x*]		Definir el valor absoluto del producto de dos expresiones arbitrarias $*x* y $$*x* como el producto de sus valores absolutos.
Abs[$*x*^($*n*_=Natp[$*n*])]:Abs[$*x*]^$*n*		Definir el valor absoluto de la expresión arbitraria $*x* elevada a la potencia del número natural $*n* como el valor absoluto de $*x* elevado a $*n*.
Abs[*a* *b*^2 *c*]	$\mathrm{Abs}[a]\ \mathrm{Abs}[b]^2\ \mathrm{Abs}[c]$	Calcular el valor absoluto del producto $a\times b^2\times c$ conforme a las reglas convencionales del álgebra y las definiciones dadas para la función valor absoluto.
Graph[Sin[E^*x*],*x*,−3,3]		Dibujar la gráfica de la función sen(e^x) para los valores de *x* comprendidos entre − 3 y + 3.

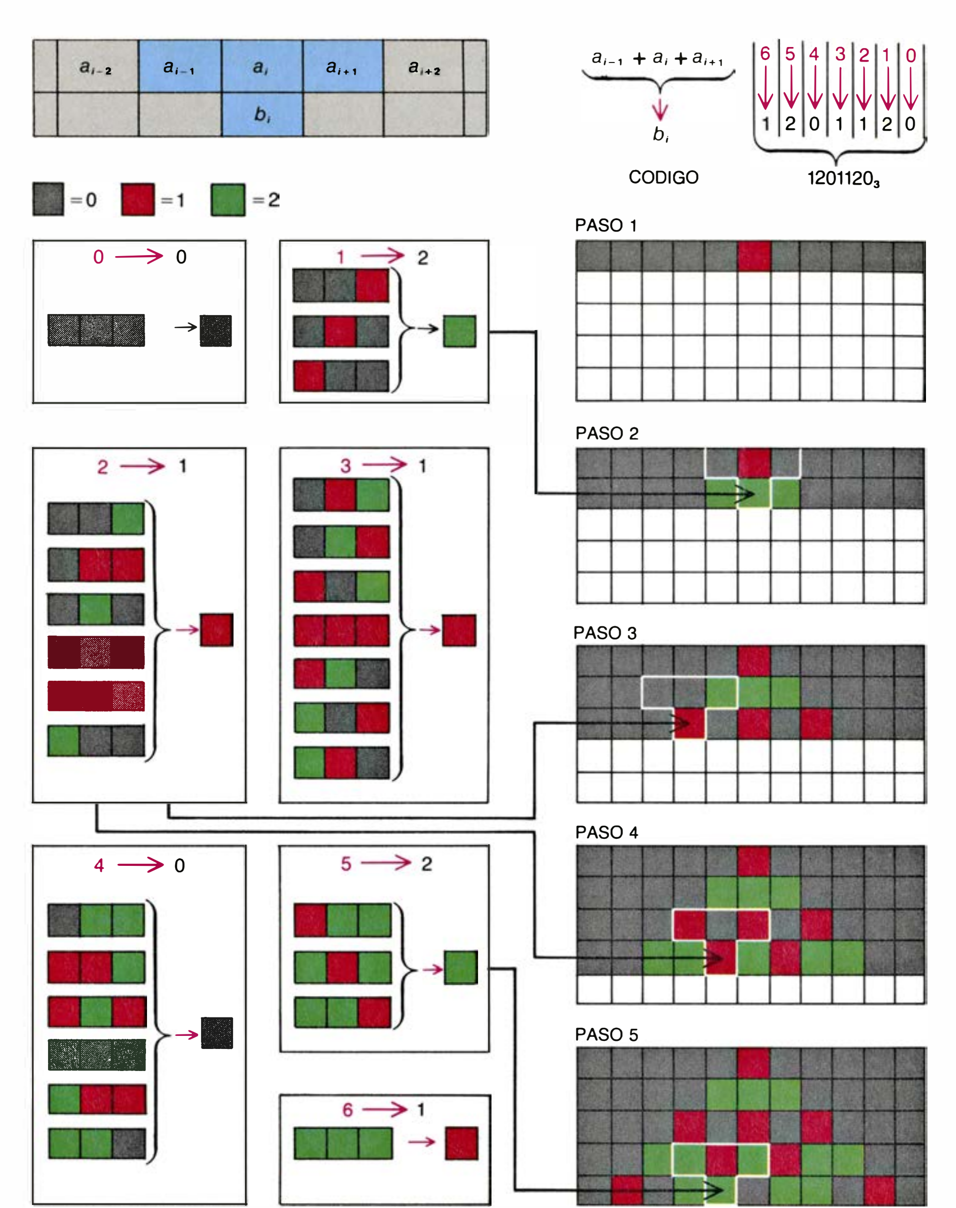
5. CALCULOS MATEMATICOS realizados en el lenguaje SMP. El computador manipula fórmulas algebraicas, otras construcciones simbólicas y números. Las instrucciones del lenguaje incluyen todas las operaciones de la matemática. Los últimos paneles indican cómo definir nuevas operaciones. Se definen las propiedades de la función valor absoluto; el ordenador las aplica luego para simplificar cualquier expresión que incluya dicha función.

simple, pero su comportamiento resulta de extrema complejidad. El estudio de tales sistemas nos conduce a un nuevo campo: la teoría de los sistemas complejos, donde los métodos computacionales desempeñan un papel central. Como caso arquetípico se puede citar la turbulencia de los fluidos que se produce, por ejemplo, cuando el agua fluye rápidamente alrededor de una obstrucción. Se formula fácilmente el conjunto de ecuaciones diferenciales que satisface el fluido. Sin embargo, las pautas de flujo de los fluidos originadas por las ecuaciones han desafiado su análisis matemático o su descripción. En la práctica, los modelos se identifican ya sea por la observación del sistema físico real, ya por el experimento con ordenador.

Se sospecha de la existencia de un conjunto de mecanismos matemáticos que sea común a muchos sistemas que muestran comportamientos complejos. Los mecanismos se estudian mejor en sistemas cuya construcción sea de la máxima sencillez posible. Tales estudios se han realizado recientemente para una clase de sistemas matemáticos conocidos por autómatas celulares. El autómata celular consta de muchos componentes idénticos; cada uno evoluciona conforme a un conjunto simple de reglas. Pero considerados en su totalidad, los componentes generan un comportamiento de complejidad esencialmente arbitraria.

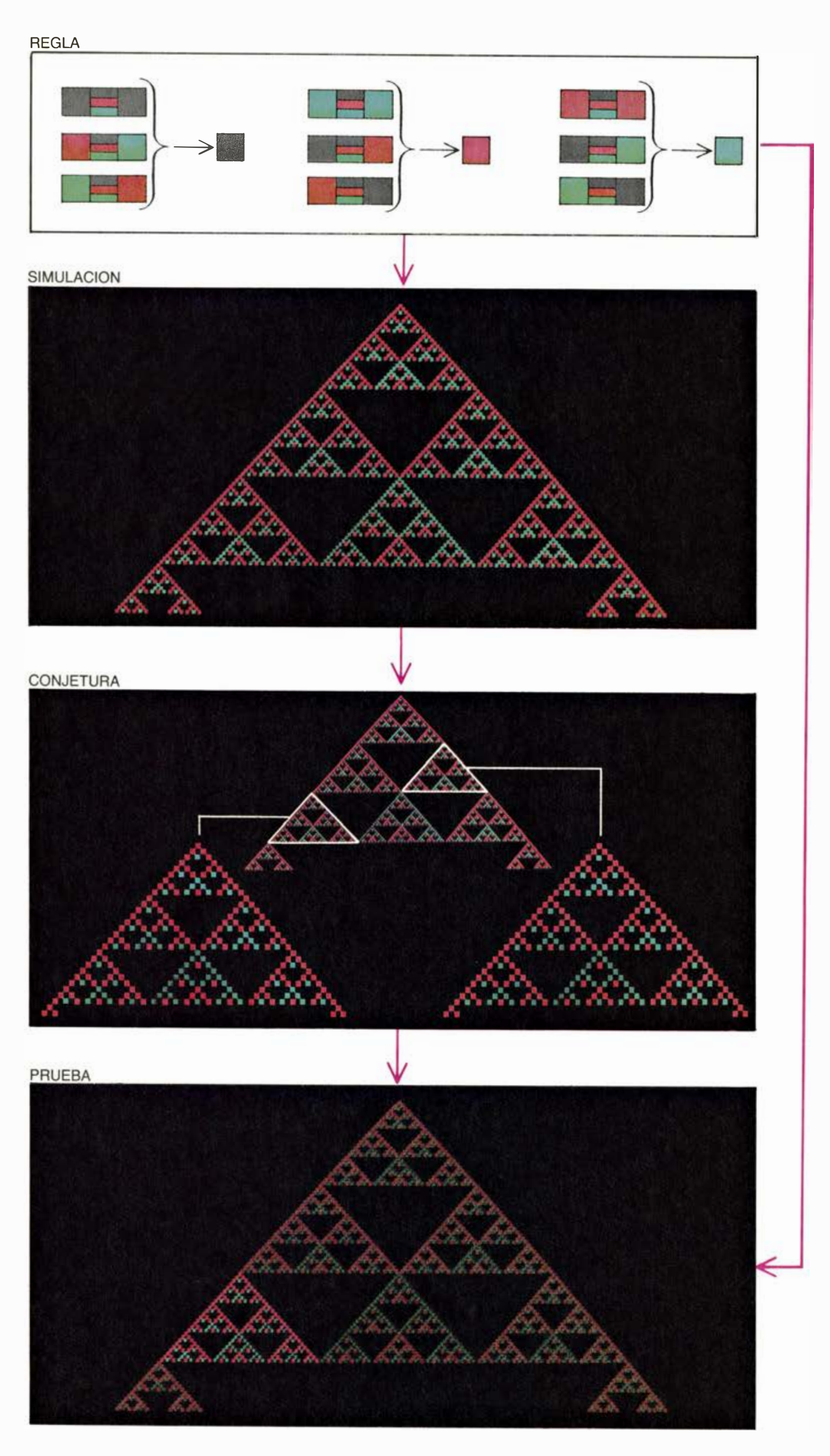
Los componentes de un autómata celular son "células" matemáticas, ordenadas en una dimensión, en una secuencia de puntos regularmente repartidos a lo largo de una línea, o en dos dimensiones como una red regular de cuadrados o hexágonos. Cada célula contiene un valor elegido dentro de un pequeño conjunto de posibilidades; muchas veces 0 o 1. Los valores de todas las células en un autómata se actualizan simultáneamente en cada "tic-tac" de un reloj, conforme a una regla predefinida. La regla fija el nuevo valor de una célula, según sus valores anteriores y los valores precedentes de sus vecinos inmediatos o de los de otro conjunto de células próximas.

Sea un autómata celular unidimensional en el cual cada célula puede tomar el valor 0 o 1. Incluso en este caso tan sencillo el comportamiento global del autómata celular puede ser muy complejo; la manera más eficaz de investigar su comportamiento es utilizar el ordenador. La mayoría de las propiedades del autómata celular se conjeturaron a partir de los modelos generados en los experimentos por ordenador. En algunos casos, se han demostrado luego con razonamientos matemáticos clásicos.



6. EN LOS AUTOMATAS CELULARES encontramos unos modelos sencillos que recogen los rasgos esenciales de múltiples sistemas naturales. El autómata celular unidimensional consta de una línea de células, mostradas en el diagrama como cuadrados coloreados. Cada célula puede tomar cierto número de valores, representados por colores diferentes. El autómata celular se desarrolla en una serie de pasos, indicados aquí en una secuencia de filas de cuadrados que avanzan de arriba abajo. A cada paso, los valores de todas las células se actualizan conforme a una regla predeterminada. En el caso aquí ilustrado, la regla especifica el nuevo valor de una célula en función de la suma de su valor anterior y del valor previo de sus vecinas inmediatas. Esas reglas vienen adecuadamente especificadas por códigos definidos como lo muestra el diagrama; el subíndice 3 indica que cada célula toma uno de tres valores posibles.

Los autómatas celulares sirven de modelos palpables para múltiples procesos físicos. Supóngase el hielo representado, en una red hexagonal bidimensional, por células de contenido 1 y el vapor de agua por células de contenido 0. Se puede entonces usar una regla de la teoría de los autómatas celulares para explicar las sucesivas etapas de la congelación de un copo de nieve. La regla dispone que, una vez congelada una célula, no se deshiela. Las células en contacto con el borde de la zona de crecimiento se hielan, a menos que tengan tanto hielo alrededor que les impida disipar el calor suficiente para congelarse. Los copos de nieve simulados por ordenador partiendo de una sola célula congelada y siguiendo esta regla presentan aspecto de árboles intricados que guardan un gran parecido con los copos de nieve reales. Un conjunto de ecuaciones diferenciales puede, también, describir el crecimiento de los copos de nieve; pero el modelo, mucho más simple, dado por el autómata celular parece preservar la esencia del proceso a través del cual se crean modelos complejos. Modelos similares explican los sistemas biológicos: esquemas com-



7. MATEMATICAS EXPERIMENTALES, una técnica exploratoria cuya viabilidad débese en gran parte, a la aparición del ordenador. La máquina puede aplicar de forma repetida cualquier conjunto de reglas matemáticas, permitiendo explorar sus consecuencias de forma experimental. Si se quiere estudiar, por ejemplo, un modelo generado por el autómata celular definido por la regla de la figura (*diagrama superior*), se empieza por simular explícitamente en un ordenador muchos pasos del desarrollo del autómata. El examen del modelo obtenido induce, entonces, la conjetura según la cual se trataría de un modelo fractal, o semejante a sí mismo, en el sentido de que sus partes, ampliadas, tienen la misma forma global que el todo resultante. La conjetura, una vez enunciada, admite una fácil comprobación por técnicas matemáticas tradicionales. La prueba se puede fundamentar en que las condiciones iniciales de crecimiento, a partir de determinadas células del modelo, son las mismas que las condiciones de crecimiento a partir de la primera célula. Los experimentos por ordenador van descubriendo nuevos resultados matemáticos. Algunos se han reproducido después mediante desarrollos matemáticos convencionales.

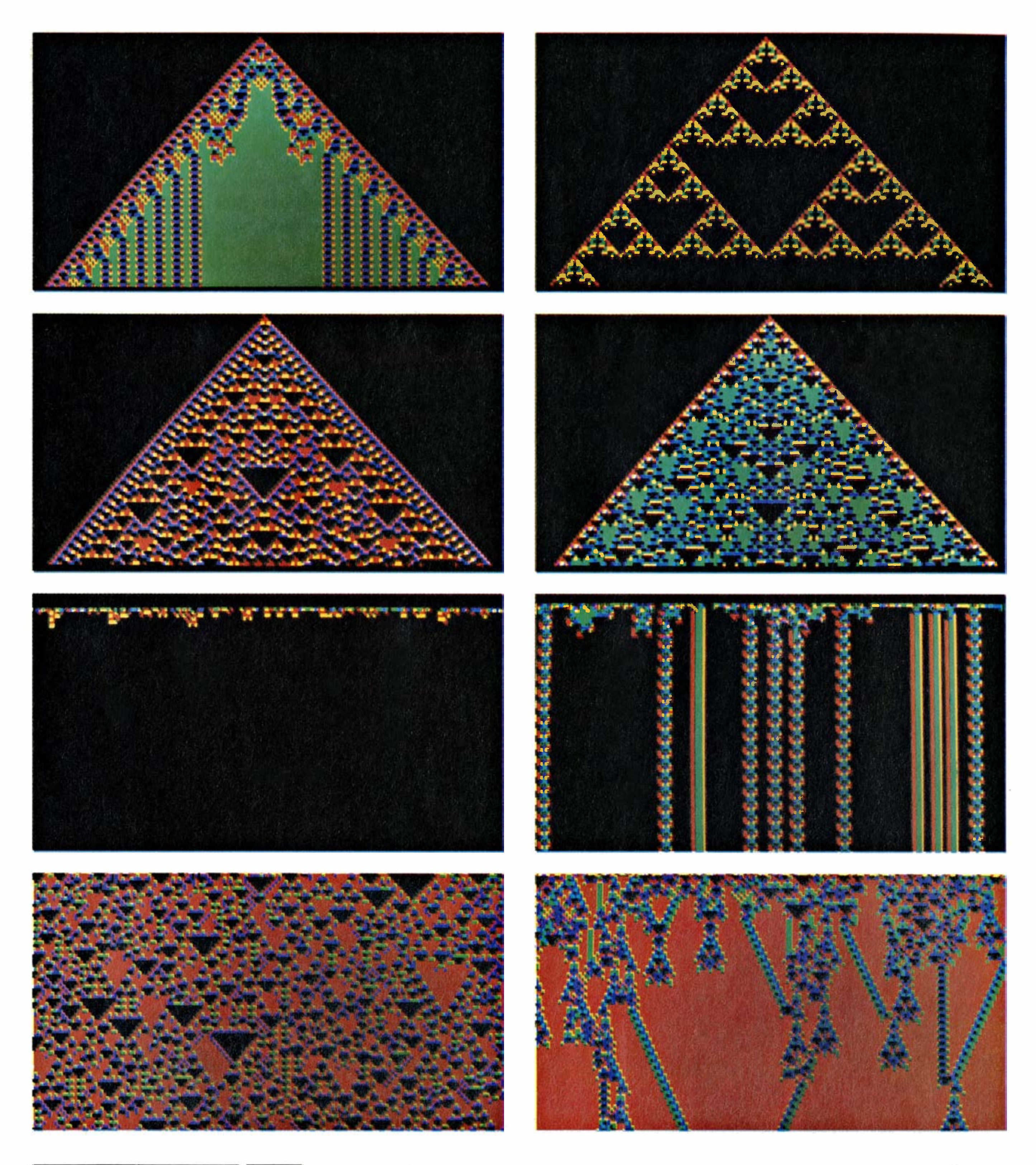
plejos de crecimiento y pigmentación pueden justificarse con algoritmos sencillos generados por el autómata celular.

La simulación por ordenador es el único método actualmente utilizado para investigar muchos de los sistemas abordados hasta aquí. Pero desde el terreno de los principios hemos de preguntar si la simulación constituye el procedimiento posible más eficiente o si hay una fórmula matemática que pudiera llevar más directamente a los resultados. Para acotar el debate, hemos de ahondar más en la correspondencia entre los procesos físicos y los procesos computacionales.

Cabe admitir por cierto que cualquier proceso físico puede describirse a través de un algoritmo, y que, por tanto, cualquier sistema físico puede representarse a través de un proceso computacional. Se ha de determinar hasta dónde llega la complejidad de este último. En el caso del autómata celular, la correspondencia entre el proceso físico y el proceso en ordenador es clara. El autómata celular puede contemplarse como un modelo para un sistema físico, pero también puede considerarse un sistema computacional análogo a un ordenador digital ordinario. En un autómata celular, la secuencia de valores iniciales de las células puede interpretarse como dato abstracto o como información, de forma muy parecida a la secuencia de dígitos binarios de un ordenador digital. A lo largo de la evolución del autómata se procesa esta información; los valores de las células se modifican según unas reglas predefinidas. De forma similar, los dígitos almacenados en la memoria del ordenador digital se modifican según las reglas contenidas en la unidad central del proceso del computador.

Así, la evolución de un autómata celular a partir de su configuración inicial se asimila a una computación que procesa la información contenida en la configuración. Para los autómatas que muestran un comportamiento simple, la computación es sencilla. Por ejemplo, puede tratarse de detectar las secuencias de tres células consecutivas cuyo valor inicial sea 1. Por otro lado, a la evolución de un autómata celular que muestra un comportamiento complicado puede corresponder una computación asimismo compleja.

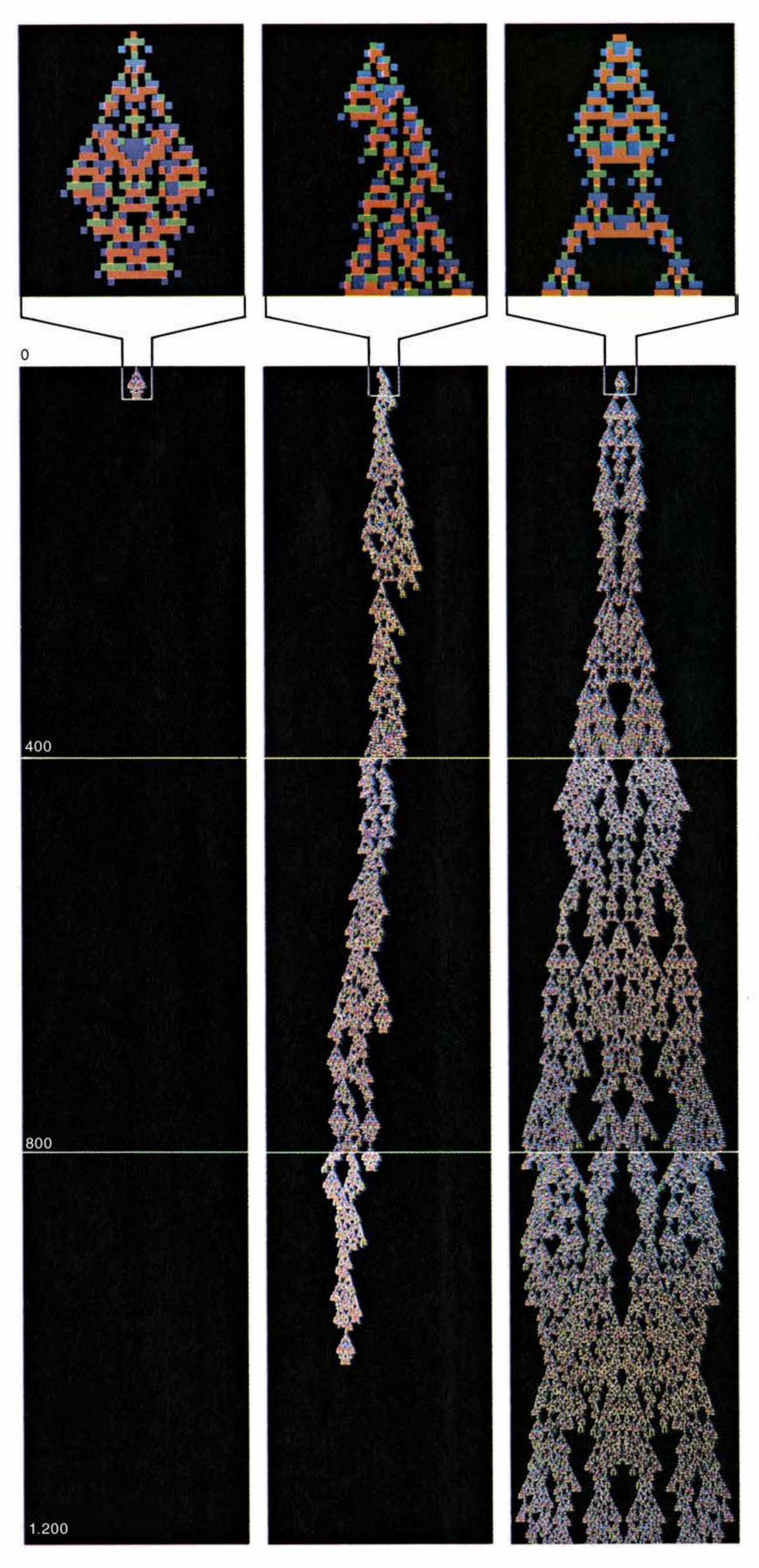
Mediante simulación explícita siempre de cada paso, podremos determinar el resultado de un número dado en la evolución del autómata celular. El problema estriba en saber si existe o no un



2213310_5	4200410_5
331240_5	2024310_5
1100400_5	2231000_5
131210_5	3211310_5

■ (negro)	= 0
■ (rojo)	= 1
■ (verde)	= 2
■ (azul)	= 3
■ (amarillo)	= 4

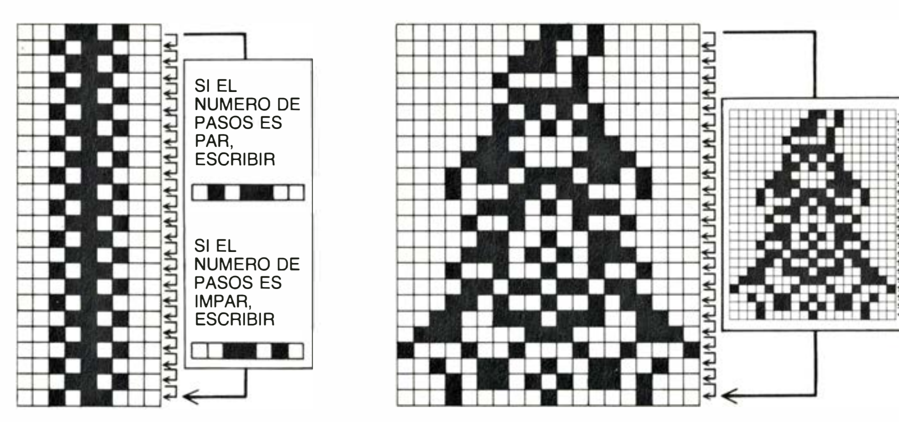
8. UN COMPORTAMIENTO COMPLEJO puede presentarse incluso en los sistemas de componentes simples. Los ocho autómatas celulares que se ofrecen en estas fotografías constan de líneas de células que toman uno de cinco valores posibles. El valor de cada célula viene determinado por una regla simple, basada en los valores de sus vecinas en la línea anterior. Cada uno de los modelos se genera por una regla cuyo código está indicado en la clave adjunta (*véase la figura 6*). Los modelos de las cuatro fotografías superiores crecen a partir de una simple célula coloreada. Incluso en este caso, los modelos generados pueden ser complejos; a veces parecen totalmente aleatorios. Los modelos complejos que se forman en el flujo turbulento de un fluido y en otros procesos físicos pueden surgir siguiendo el mismo mecanismo. Los modelos complejos generados por un autómata celular sirven también de fuentes eficaces de números aleatorios; se pueden aplicar también para cifrar mensajes, transcribiendo un texto en otro aparentemente aleatorio. Los modelos de las cuatro fotografías inferiores empiezan en estados desordenados. Aunque los valores de las células de los estados iniciales estén escogidos al azar, la evolución del autómata celular lleva a estructuras de cuatro clases básicas. En las dos clases que se ven en la tercera línea de fotografías, el comportamiento a largo plazo del autómata celular es bastante sencillo; en los dos casos mostrados en la última línea, puede resultar bastante complejo. A buen seguro, muchos sistemas naturales se comportarán según la clasificación expuesta.



procedimiento más eficaz. ¿Existe un ataque que acorte la simulación paso-a-paso, un algoritmo que dé el resultado de muchos pasos en la evolución de un autómata celular sin tener que seguirlos todos? El ordenador podría ejecutar este algoritmo y sería posible predecir la evolución de un autómata celular sin simularlo explícitamente. El fundamento de esta operación consistiría en que el ordenador pudiese llevar a cabo una computación más refinada que la que realiza el autómata celular, y alcanzar así los mismos resultados en menos pasos. Para decirlo plásticamente: sería como si el autómata celular calculara 7 veces 18 sumando siete veces 18, en tanto que el computador obtendría el mismo producto recurriendo a la multiplicación. Tal atajo es posible solamente si el ordenador logra realizar unos cálculos intrínsecamente más complicados que los involucrados en la evolución del autómata celular.

Se puede definir una cierta clase de problemas, llamados problemas calculables, que son los que admiten solución en un tiempo finito siguiendo unos algoritmos determinados. Un ordenador simple, piénsese en una máquina sumadora, puede resolver sólo un pequeño subconjunto de esos problemas. Pero existen ordenadores universales, de uso general, que resuelven cualquier problema calculable. Los ordenadores digitales son máquinas universales. Las instrucciones que puede ejecutar el procesador central del ordenador son lo bastante ricas como para servir de elementos a un programa que pueda incorporar cualquier algoritmo. Además de los computadores digitales, cierto número de sistemas se han mostrado capaces de computación universal. Citemos, entre otros, algunos autómatas

9. PROBLEMAS INDECIDIBLES surgen en el análisis matemático de los modelos de ciertos sistemas físicos. Tratemos de determinar, por ejemplo, si un modelo generado en el desarrollo de un autómata celular llegará a extinguirse y tener todas las células negras. Los modelos generados por el autómata arriba mostrado son tan complicados que el único enfoque general posible para buscar la solución del problema es simular, explícitamente, la evolución del sistema celular. El modelo obtenido a partir del estado inicial, que se ve a la izquierda, se agota en 16 pasos. El estado inicial del centro produce un modelo que tarda 1016 pasos en extinguirse. El estado inicial de la derecha origina un modelo cuyo final queda incierto, después incluso de la simulación de muchos millares de pasos. En general, no hay ninguna garantía de que la simulación finita de un número fijo de pasos pueda determinar el comportamiento final del autómata celular. Por eso, el problema de saber si un modelo particular acabará o no por extinguirse, o si se detendrá, se denomina formalmente indecidible. El autómata celular mostrado aquí sigue una regla especificada por el número de código 3311100320_4.



10. IRREDUCCION COMPUTACIONAL, fenómeno que parece producirse en muchos sistemas físicos y matemáticos. Si se quiere averiguar el comportamiento de un sistema basta con simular uno a uno los pasos de su desarrollo. No obstante, cuando es lo suficientemente simple, cabe dar con un atajo: una vez conocido su estado inicial, el que ostente en cualquiera de los pasos siguientes puede deducirse directamente con una fórmula matemática. Para el sistema mostrado esquemáticamente a la izquierda, la fórmula requiere el cálculo del resto de la división por dos del número de pasos. Un sistema así se dice computacionalmente reducible. Para un sistema tal como el mostrado esquemáticamente a la derecha, sin embargo, el comportamiento es tan complicado que, en general, no se puede dar ninguna descripción resumida de su evolución. Tal sistema es computacionalmente irreducible, y su evolución se puede determinar realmente sólo a través de la simulación de cada paso. Parece probable que muchos sistemas físicos y matemáticos, para los cuales no se conoce ninguna descripción simple, son, de hecho, computacionalmente irreducibles. No queda otra forma de estudiarlos que el experimento, sea físico o computacional.

celulares; en este sentido, se ha probado la capacidad de computación universal de un simple autómata celular bidimensional con 0 o 1 en cada célula. Razones poderosas inducen a pensar que varios autómatas celulares unidimensionales son también ordenadores universales. Los candidatos más simples tienen tres posibles valores para cada célula y reglas de evolución que toman en cuenta solamente las células más cercanas.

Los autómatas celulares capaces de computación universal imitan el comportamiento de cualquier ordenador posible. Y dado que cualquier proceso físico puede representarse como un proceso computacional, pueden también imitar la acción de cualquier sistema físico posible. Si hubiera un algoritmo capaz de seguir el comportamiento de esos autómatas celulares con una celeridad mayor que la propia evolución de los autómatas celulares, permitiría acelerar cualquier computación. Puesto que esta conclusión lleva a una contradicción lógica, se deduce que no puede haber un atajo válido general para predecir la evolución de un autómata celular arbitrario. Los cálculos correspondientes a la evolución son irreducibles: el resultado sólo puede obtenerse eficazmente a través de la simulación explícita de la evolución. De hecho, esta simulación directa constituye el mejor método para determinar el comportamiento de ciertos autómatas celulares. No hay manera de predecir su evolución; sólo queda observar lo que pasa.

No se sabe todavía cuán extendido se halla el fenómeno de la irreducción computacional entre los autómatas celulares, ni entre los sistemas físicos en general. A pesar de lo cual está claro que los elementos de un sistema no necesitan ser muy complicados para que la evolución global del mismo sea computacionalmente irreducible. Cabe incluso que la irreducción computacional se dé casi siempre que el comportamiento de un sistema se muestre complejo o caótico. No se conocen fórmulas matemáticas generales que describan el comportamiento global de esos sistemas; quizá no lleguen a encontrarse nunca fórmulas así. En tal caso, la simulación explícita en el ordenador es el único método de investigación disponible.

Tradicionalmente, la ciencia física se ha centrado, en su mayor parte, sobre el estudio de los fenómenos computacionalmente reducibles, que admiten una descripción simple y global. En los sistemas físicos reales, no obstante, la reducción computacional es más la excepción que la regla. La turbulencia de los fluidos es probablemente uno de los numerosos ejemplos de irreducción computacional. En los sistemas biológicos, la extensión de la irreducción computacional puede ser todavía mayor: quizás ocurriera que la forma de un organismo biológico pudiera determinarse a partir de su código genético, sin más que seguir, paso a paso, su desarrollo. En los casos de irreducción computacional, se ha de adoptar una metodología que depende estrictamente de la computación.

De la irreducción computacional dedúcese que hay preguntas que pueden hacerse sobre el comportamiento fundamental de un sistema, pero a las cuales no se puede contestar de forma general mediante un proceso matemático o computacional finito. Tales cuestiones, por tanto, han de considerarse indecidibles. Demos un ejemplo: averiguar si un modelo particular acabará por extinguirse en la evolución de un autómata celular. Esta cuestión es fácil de contestar para un número dado de pasos, 1000, por ejemplo; solamente hay que simular 1000 pasos en la evolución del autómata celular. No obstante, a fin de conocer la respuesta para cualquier número de pasos, se ha de simular la evolución del autómata celular para un número de pasos potencialmente infinito. Si el autómata celular es computacionalmente irreducible, no hay alternativa real a esta simulación directa.

La conclusión es que no hay ningún proceso de cálculo de longitud fija que pueda determinar categóricamente si un modelo se extinguirá al final. Se puede encontrar el destino de un modelo particular siguiendo sólo unos pocos pasos en su evolución, pero no hay forma general de conocer de antemano cuántos pasos se necesitarán. La forma final de un modelo es el resultado de un número infinito de pasos, correspondiente a una computación infinita; a menos que la evolución del modelo sea computacionalmente reducible, sus consecuencias no pueden reproducirse con ningún proceso computacional o matemático finito.

La posibilidad de cuestiones indecidibles en los modelos matemáticos de los sistemas físicos puede verse como una manifestación del teorema de Gödel sobre la indecidibilidad en matemáticas, demostrado por Kurt Gödel en 1931. El teorema establece que en todos los sistemas matemáticos, hasta en los más simples, caben proposiciones que no pueden ni probarse ni refutarse con un proceso matemático o lógico finito. La prueba de una proposición puede requerir un número de pasos lógicos indefinidamente grande. Incluso proposiciones de formulación concisa pueden exigir una prueba arbitrariamente larga. En la práctica, hay muchos teoremas matemáticos simples para los cuales las únicas demostraciones conocidas son muy largas. Además, los casos que han de examinarse para probar o refutar conjeturas son, a menudo, muy complicados. En la teoría de números, por ejemplo, hay muchos casos en que el número más pequeño que posee una propiedad especial dada es extremadamente grande; a menudo,

este número sólo puede encontrarse tanteando de uno en uno. Fenómenos de esta índole convierten al ordenador en una herramienta esencial en muchas investigaciones matemáticas.

La irreducción computacional implica muchas limitaciones fundamentales sobre el alcance de las teorías para los sistemas físicos. Quizá fuera posible diseñar modelos de un sistema contemplado en muchos niveles, desde simular los movimientos de las moléculas, una a una, hasta resolver las ecuaciones diferenciales para las propiedades globales. La irreducción computacional implica la existencia de un tope superior para los modelos abstractos que puedan construirse. Rebasado ese tope sólo cabe encontrar resultados por simulación explícita.

Cuando el nivel de descripción alcanza la irreducción, comienzan a aparecer también cuestiones indecidibles. Este tipo de cuestiones se han de evitar en la formulación de una teoría, de la misma manera que la medida simultánea de la posición y de la velocidad de un electrón –imposible conforme al principio de incertidumbre– se evita en la mecánica cuántica. Aun eliminando esas cuestiones, queda la dificultad de contestar a las preguntas que, en principio, tienen respuesta. El grado de dificultad depende fundamentalmente de la naturaleza de los objetos involucrados en la simulación. Si el único camino para predecir el tiempo fuera simular el movimiento de cada molécula de la atmósfera, no se podría realizar ningún cálculo práctico. Sin embargo, los rasgos relevantes del tiempo pueden estudiarse, probablemente, considerando las interacciones de grandes masas de la atmósfera; y así es como nacen simulaciones útiles.

La fiabilidad e interés con que un sistema irreducible puede simularse dependen del refinamiento computacional de cada paso de su evolución. Las fases de evolución del sistema pueden simularse con instrucciones de un programa. Cuantas menos instrucciones se necesiten para reproducir un paso, más eficiente será la simulación. Las descripciones de un sistema físico a alto nivel exigen pasos complejos, a imagen de lo que ocurre con las instrucciones únicas en lenguajes de alto nivel, que corresponden a muchas instrucciones en lenguajes de niveles inferiores. Un paso en la aproximación numérica de una ecuación diferencial que describe un chorro de gas exige una computación más refinada que la requerida para seguir el choque de dos moléculas en un gas. Por otra parte, cada paso de la descripción a alto nivel dada por la ecuación diferencial equivale a un inmenso número de pasos de la descripción a bajo nivel de las colisiones moleculares. La ganancia en rendimiento resultante compensa, con creces, la complejidad mayor de los distintos pasos singulares.

En general, el rendimiento de una simulación va a la par con el nivel de descripción, hasta que las operaciones exigidas por la descripción a alto nivel se ajusten a las operaciones obtenidas directamente por el ordenador encargado de la simulación. Lo deseable sería que la computación guardara una estrecha semejanza con el sistema simulado.

Hay una diferencia principal. Entre la mayoría de los ordenadores y los sistemas físicos o sus modelos media una diferencia sustancial: los ordenadores procesan la información en serie, en tanto que los sistemas físicos procesan la información en paralelo. En el sistema físico descrito por un autómata celular, los valores de todas las células se actualizan al mismo tiempo y a cada paso. En un programa convencional, la simulación del autómata celular la realiza un bucle, que actualiza el valor de cada célula, una por una. En cuyo caso, resulta fácil escribir un programa que realice un proceso fundamentalmente paralelo con un algoritmo en serie. Hay un sistema bien establecido, en cuyo marco pueden escribirse algoritmos de procesos en serie. Muchos sistemas físicos parecen, no obstante, requerir descripciones esencialmente paralelas por naturaleza. No existe todavía una sistematización de los problemas paralelos, pero cuando se desarrolle permitirá descripciones más eficientes a alto nivel de los procesos físicos.

La introducción del computador en las ciencias es un fenómeno reciente. Pero el tratamiento por ordenador permite abordar, con nuevos planteamientos, muchos problemas. Posibilita el estudio de fenómenos mucho más complejos que los que se podían considerar antes y está cambiando la dirección y el acento de muchos campos de la ciencia. Y lo más importante, posiblemente: introduce una nueva forma de pensar en las ciencias. Las leyes científicas se contemplan ahora con algoritmos. Muchas se estudian en experimentos con computador. Los sistemas físicos han pasado a considerarse sistemas computacionales, que procesan la información a la manera del computador. Nuevos aspectos de los fenómenos naturales se han hecho accesibles a la investigación. Un nuevo paradigma ha nacido.

Programación de sistemas inteligentes

La clave para la resolución inteligente de problemas reside en reducir la búsqueda aleatoria. Los programas inteligentes deberán acudir, por tanto, a las mismas "fuentes de poder" que utiliza el hombre

Douglas B. Lenat

Una tarde de verano, camino de Portugal, llega el lector a un cruce en Valladolid. La carretera sigue hacia delante, en línea recta. A la izquierda, la que cruza deja su estela a través de los campos de trigo; a la derecha, protegiéndose los ojos del sol, se contempla el mismo panorama. Sin ayuda de un mapa ni de otra indicación, decide escoger al azar una de las tres y gira, por ejemplo, a la izquierda. Pronto llega a otra intersección, y luego a otra, lo que le obliga a tomar una serie de decisiones al azar; en más de una ocasión se encuentra en una carretera sin salida, da media vuelta y ensaya otra. Sólo una larga vida y una envidiable suerte permitirían al lector llegar a Portugal, pero la probabilidad de que lo logre es sólo de uno en 10^{30}. En realidad, sin embargo, puesto que algo sabe de geografía, el lector no avanza por la campiña al azar y, a la primera intersección, toma el camino correcto.

Muchos problemas, la mayoría bastante más interesantes que el anterior, responden al mismo modelo: la búsqueda de un camino desde el estado inicial del problema hasta su solución, o estado final, deseada. La mayoría de los problemas interesantes comparten también con el anterior la característica de ser demasiado complejos para resolverlos buscando al azar el camino que lleva a la solución, ya que el número de elecciones a realizar crece exponencialmente a medida que se avanza desde la primera intersección, o punto de decisión. Un ejemplo clásico lo tenemos en el ajedrez, cuyo número de posiciones permitidas sobre el tablero se ha estimado en unas 10^{120}. Sin embargo, un buen jugador reduce el problema a proporciones manejables y, al escoger su siguiente jugada, considera sólo unas 100 posiciones, las correspondientes a las líneas de ataque más prometedoras. Y en eso consiste, en opinión del autor, la esencia de la inteligencia: encontrar procedimientos que, limitando la búsqueda de soluciones, resuelvan problemas que de otra forma resultarían intratables.

Durante casi 30 años, una pequeña comunidad de investigadores ha intentado, con éxito diverso, programar ordenadores para hacer de ellos sistemas inteligentes de resolución de problemas. A mitad de los años setenta, después de dos décadas de progreso tediosamente lento, los investigadores de ese nuevo campo de la inteligencia artificial llegaron a una conclusión fundamental sobre el comportamiento inteligente en general: requiere una tremenda cantidad de conocimientos, que la gente da a menudo por supuestos, pero que deben aportarse al ordenador con todo detalle. Situado en la citada intersección de Valladolid, el viajero sabría que Portugal queda al oeste, que por la tarde el sol está en el oeste y deduciría, por tanto, que, situándose con el sol de frente, estaría en la buena dirección. Así, se ahorraría el ensayo de las otras dos carreteras.

La relativa simplicidad de este problema, sin embargo, no es representativa de otras tareas cotidianas que la gente resuelve sin pensarlo dos veces. Por ejemplo, entender las expresiones del lenguaje corriente, incluso las más sencillas, requiere amplios conocimientos del contexto, sobre quién habla y el mundo en general, que están fuera de la capacidad de los programas de ordenador actuales. El papel central que corresponde al conocimiento en la inteligencia explica por qué los programas de ordenador con más éxito han sido los llamados "sistemas expertos", los cuales operan en dominios altamente especializados, como el diagnóstico de la meningitis o los programas de juegos. En cambio, al realizar los primeros esfuerzos para diseñar un "resolutor general de problemas" se dio por supuesto que el núcleo de la inteligencia residía en una capacidad de razonar aplicable a todos los dominios. Esos esfuerzos resultaron poco fructíferos y se han abandonado en su mayor parte.

Al afrontar un problema complejo, y para limitar la búsqueda de una solución, la gente se vale de diversos métodos, que yo llamo fuentes de poder, fundamentados en su conocimiento de las regularidades generales. Se pueden invocar teoremas matemáticos o reglas empíricas no tan formales; se puede descomponer el problema inicial en subproblemas de manejo más cómodo, o cabe razonar por analogía con problemas ya resueltos. Los programas de ordenador ponen de manifiesto inteligencia en la medida en que se basan en al-

1. PROGRAMA INTELIGENTE pasado por una máquina que muestra varios aspectos de su funcionamiento en diferentes ventanas. El programa, llamado EURISKO, escrito por el autor y sus colegas, se ha aplicado a cierto número de tareas, incluidas las relacionadas en la ventanita inferior rotulada "Topics"; en este ejemplo el programa se ocupa de proyectar una flota para jugar al juego de guerra Traveller T.C.S. La base de conocimientos del programa contiene las complejas reglas del juego, así como otros criterios heurísticos generales para guiar la búsqueda de diseños cada vez mejores. En la figura, EURISKO ha finalizado la simulación de una batalla en la que el "bando 1" ha derrotado sin paliativos al "bando 2"; el planteamiento heurístico en curso le permite aprender la razón de la victoria, a partir de los resultados de la batalla, mediante el análisis de las diferencias entre las dos flotas. Puesto que la principal diferencia reside en que el "bando 1" sólo posee un tipo de barcos, EURISKO plantea la hipótesis de que es deseable minimizar el número de tios de barcos y, al mismo tiempo, sugiere un experimento que ponga a prueba la hipótesis. La flota de EURISKO ha derrotado a menudo formaciones diseñadas por humanos.

Prompt Window

This task is taking a long time;
type a few ↑T's if you would
like me to move on to a new one.

Current-Agenda

NumberOfTasks: 1307

(RunMore TravellerFleetBattle)
7 Reasons
(Analyze Battle821)
7 Reasons
(Analyze Battle815)
2 Reasons
(Mutate ShipType37 (Increase Agility))

7 Reasons
(Analyze Battle816)
5 Reasons
(Mutate TicTacToe (Increase Complexity

Top level typescript window

Beginning task 459-110:
Analyze the DifferenceBetween
side1 and side2 in the recent
TravellerFleetBattleGame played,
looking for Cause.

The main difference is that
side1 has ships of one type,
while side2 has ships of nine
types. 4 other hypotheses.

1 is more likely to be special
than 9. So consider: If
designing a fleet for TFBG,
minimize the types of ship.

Experiment: reduce side2's
number of ships, and increase
side1's number of ship types.

Current-Concept

Worth: 613
NPlayers: 2
InitialWorth: 650
ToPlay: (PlayTravellerFleetBattle)
Rules: (TravelleRules
RulesOfFairPlay)
IsA: (Game Anything TwoPersonGame
FairGame WarGame FleetBattle
DiceGame)

... at the moment ...

Adding a task to the Agenda,
to Analyze the differences
(i.e., between side1 and
side2) for Cause.

Current-Heuristic

H161: (IF the current task was to find an Applic of a
Game, THEN try to learn from the results)

*** CONDITIONS ***
IfPotentiallyRelevant: (Playing (a Game))
IfFinishedWorkingOnTask: (a GamePlaying)
IfResultsSatisfied: (a Decisive Victory)

*** ACTIONS ***
ThenCompute: (DifferenceBetween side1 side2)
ThenPrintToUser: (Guessed the causes in the recent --
)
ThenAddToAgenda: (Analyze the differences for Cause)

*** DESCRIPTIONS ***
IsA: (Heuristic Op Anything MultiValuedOp AbstractOp)
Worth: 542
Abbrev: (It's worth finding out why one --)
Arity: 1
InitialWorth: 400
LastRunOn: PlayTravellerFleetBattle
ThenAddToAgendaRecord: (7560 . 6)
ThenPrintToUserRecord: (6726 . 21)
OverallRecord: (351537 . 21)
ThenComputeFailedRecord: (501 . 1)
ThenComputeRecord: (314972 . 15)
Generalizations: (ProtoOp)
FocusTask: (FocusOnH61)

TOPICS

ElemMathematics
Heuristics
Representation
OilSpills
Programming
Games
DevicePhysics
Plumbing
PlaneTessellation

Current-Task

RunMore-TravellerFleetBattle
:

Priority: 873
IsA: (Task SimulationTask
GameTask)
ConceptToWorkOn:
TravellerFleetBattle
AspectToWorkOn: Play
CreditTo: (Heuristic85
Heuristic13 TheUser)
PastHistory: ((Run112 Task 1203
Created ShipType30 ShipType31
& 10 others)))
NReasons: 7
Reasons: ((Because it is a
valuable concept) (Because the
user is interested in it) (
Because in the past it led to
useful new --) (Because there
is not much else interesting to
do --))

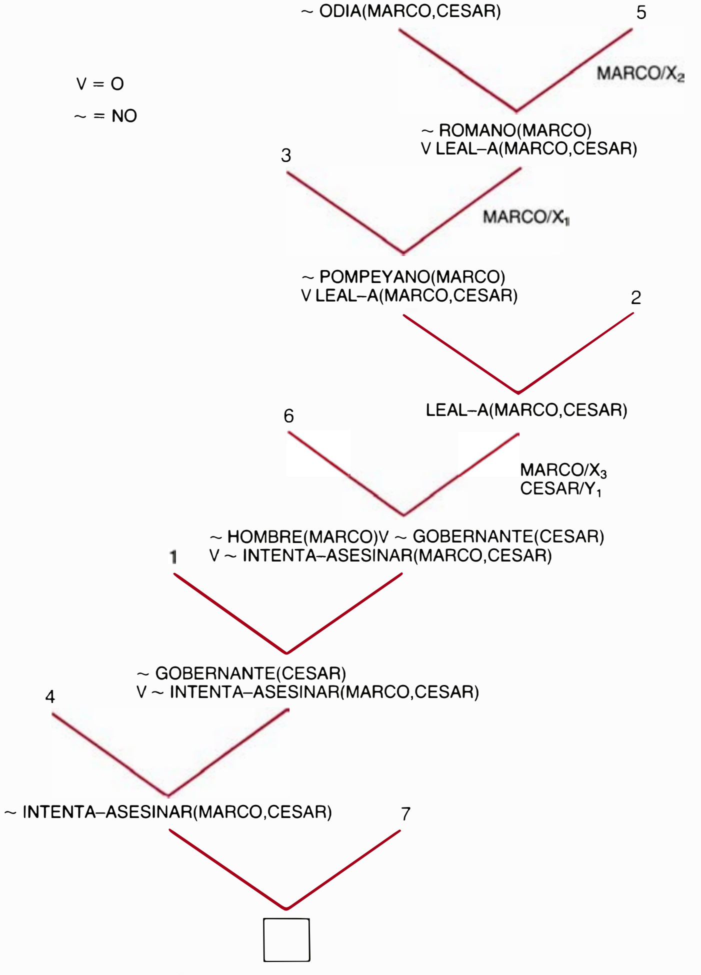
Plus 4
properties which are not slot names:
(NReasons PastHistory

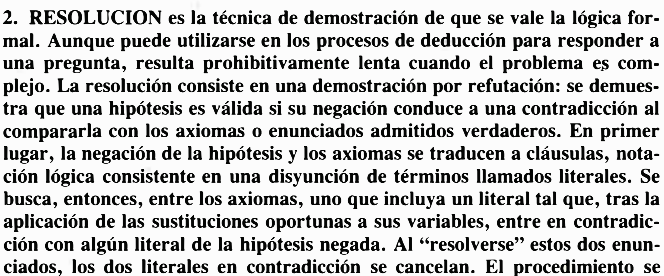
gunas de esas mismas fuentes de poder. El futuro de la inteligencia artificial depende de que se encuentren vías de conexión con aquellas fuentes cuya explotación no ha hecho sino empezar.

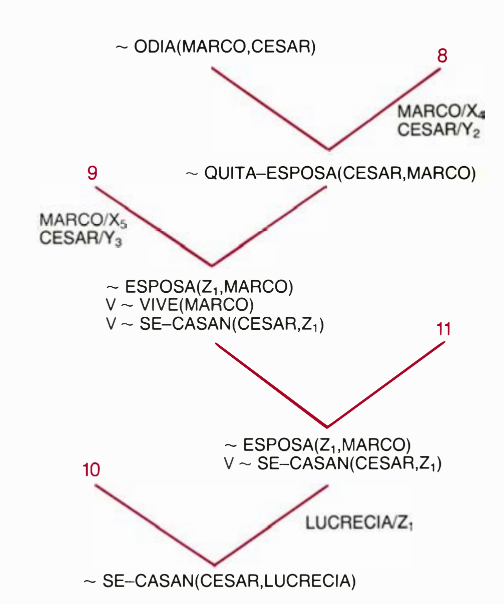
Muchos de los programas escritos durante las dos primeras décadas de investigación en inteligencia artificial dependían, en exceso, de los métodos de razonamiento formal. Tales métodos proporcionan procedimientos poderosos para podar, o incluso eliminar, el árbol de búsqueda, siempre que el problema a resolver esté bien definido y su dominio sea muy restringido. Por ejemplo, nadie necesita malgastar tiempo en la búsqueda de una solución al problema de la trisección de un ángulo, o en la búsqueda del mejor método para calcular el movimiento de un proyectil, puesto que ya se han creado teoremas y algoritmos bien fundados que han resuelto esas cuestiones de una vez por todas.

"¿ODIABA MARCO A CESAR?"

1	HOMBRE(MARCO)	Marco era un hombre.
2	POMPEYANO(MARCO)	Marco era un pompeyano.
3	~ POMPEYANO(X_1)VROMANO(X_1)	Todos los pompeyanos eran romanos.
4	GOBERNANTE(CESAR)	César era un gobernante.
5	~ ROMANO(X_2)VLEAL-A(X_2,CESAR)VODIA(X_2,CESAR)	Todos los romanos eran leales a César o le odiaban.
6	~ HOMBRE(X_3)V ~ GOBERNANTE(Y_1)V ~ INTENTA–ASESINAR(X_3,Y_1)V ~ LEAL-A(X_3,Y_1)	Sólo se intenta asesinar a gobernantes a los que no se es leal.
7	INTENTA–ASESINAR(MARCO,CESAR)	Marco intentó asesinar a César.
8	~ QUITA–ESPOSA(Y_2,X_4)VODIA(X_4,Y_2)	Una persona odia a aquel que le quita su esposa.
9	~ ESPOSA(Z_1,X_5)V ~ VIVE(X_5)V ~ SE–CASAN(Y_3,Z_1)VQUITA–ESPOSA(Y_3,X_5)	Si la esposa de un hombre vivo se casa con otro hombre, éste le quita la esposa al primero.
10	ESPOSA(LUCRECIA,MARCO)	Lucrecia era la esposa de Marco.
11	VIVE(MARCO)	Marco estaba vivo.







2. RESOLUCION es la técnica de demostración de que se vale la lógica formal. Aunque puede utilizarse en los procesos de deducción para responder a una pregunta, resulta prohibitivamente lenta cuando el problema es complejo. La resolución consiste en una demostración por refutación: se demuestra que una hipótesis es válida si su negación conduce a una contradicción al compararla con los axiomas o enunciados admitidos verdaderos. En primer lugar, la negación de la hipótesis y los axiomas se traducen a cláusulas, notación lógica consistente en una disyunción de términos llamados literales. Se busca, entonces, entre los axiomas, uno que incluya un literal tal que, tras la aplicación de las sustituciones oportunas a sus variables, entre en contradicción con algún literal de la hipótesis negada. Al "resolverse" estos dos enunciados, los dos literales en contradicción se cancelan. El procedimiento se aplica sucesivamente al enunciado resultante de la resolución de dos enunciados; finalmente, si la hipótesis original era válida, el proceso termina con una contradicción a secas. En el ejemplo presentado, la hipótesis es "Marco odiaba a César". En el caso imaginario de la izquierda se dispone de poca información; sólo un axioma (*5*) incluye un literal en contradicción con la negación de la hipótesis, de modo que un programa de ordenador completa rápidamente la demostración. Cuando disponemos de más información (*axiomas en color*), por ejemplo una nueva causa de odio (*8*), el programa quizás escoja el axioma menos conveniente y llegue a un punto muerto sin generar la contradicción (*derecha*). En los problemas del mundo real, el número de elecciones a realizar crece exponencialmente con el número de axiomas, que es muy grande, con lo cual resulta irrealizable una búsqueda a ciegas de la solución.

Uno de los métodos formales más populares ha sido el de deducción lógica, que utiliza una técnica de demostración por refutación, llamada resolución. Su aplicación exige, en primer lugar, convertir el enunciado a demostrar en una fórmula lógica del cálculo de predicados. Se niega, entonces, este enunciado y su negación se "resuelve" con una serie de axiomas, que son los enunciados tenidos por verdaderos en el área particular tratada. Si en las inferencias obtenidas combinando la negación con los axiomas surge una contradicción, la negación debe ser falsa y, por tanto, verdadero el enunciado original.

En 1964, J. A. Robinson demostró que el método de resolución es "completo": generará siempre una contradicción si el teorema a demostrar es verdadero. Si el teorema es falso, no se puede garantizar que acabe el proceso de inferencia. El trabajo de Robinson dio pie a una década de aplicaciones de la resolución y de otros métodos formales relacionados con la demostración automática de teoremas. Puso de manifiesto que los programas de ordenador pueden demostrar teoremas de dificultad moderada, y que el método de resolución puede adoptarse también en programas cuyo propósito es más contestar preguntas que demostrar teoremas. El gran defecto del método de resolución es estar sujeto a la "explosión combinatoria": el número de resoluciones que el programa ha de ensayar crece exponencialmente con la complejidad del problema. Progamas que aplican con éxito la resolución de pequeños casos fallan al abordar problemas más interesantes del mundo real.

La misma dificultad acosa a los programas de ordenador que se sirven de otra técnica lógica, la llamada inducción estructural. A esos programas se les suministra gran cantidad de datos sobre los objetos de un determinado dominio y se les pide construir un árbol de decisión que discrimine unos objetos de otros. Pero los algoritmos de inducción estructural presentan un problema: no incorporan ninguna información que les permita decidir qué variables son las importantes, o tratar datos que crean ruido o casos excepcionales. Si en un dominio dado el número de objetos y de características asociadas es grande, el árbol de decisión generado por el programa puede resultar desmesurado.

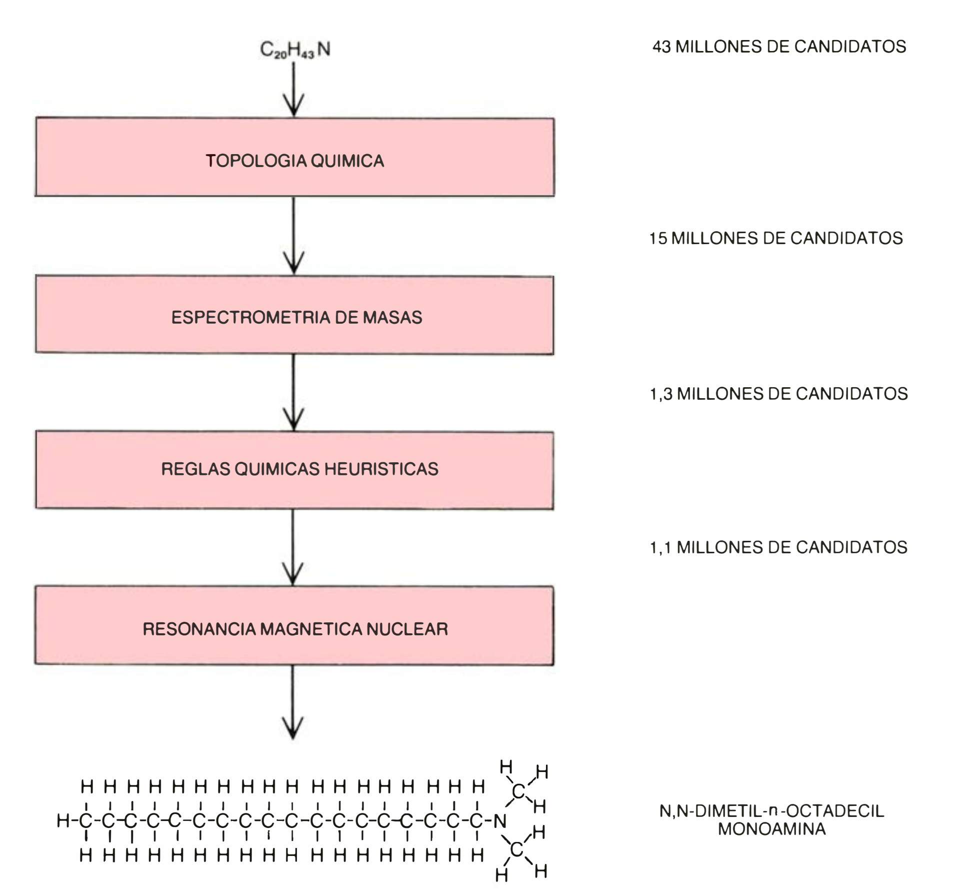
Para que el razonamiento formal rinda bien como única fuente de poder de un programa, el problema a tratar ha de ser pequeño. Una de las aplicaciones fructíferas del razonamiento formal en un futuro próximo puede ser la simulación del razonamiento cualitativo en física. John Seely Brown y Johan de Kleer, ambos del Centro de Investigación de la Xerox en Palo Alto, han escrito un programa que describe las fluctuaciones de presión en una válvula reguladora mediante ecuaciones cualitativas. Si le decimos al programa, por ejemplo, que la presión en el lado izquierdo de la válvula ha aumentado, las ecuaciones se alteran convenientemente y el programa predice el cambio de presión al otro lado de la válvula, así como el estado de equilibrio resultante en el sistema. Si bien éste es un ejemplo muy sencillo, se está usando un enfoque similar en el análisis y el diseño de circuitos eléctricos.

Sin embargo, muchos problemas interesantes no pueden resolverse con la mera ayuda del razonamiento formal. El poder de los métodos lógicos reside en su representación del mundo mediante símbolos, cuya manipulación por procedimientos bien conocidos (la resolución, por ejemplo) permite elaborar inferencias. Ese poder constituye también su mayor debilidad: muchos tipos de conocimiento, entre ellos el incierto e incompleto tan característico de la mayoría de los problemas del mundo real, no se prestan a representación por formalismos lógicos precisos. Los programas basados exclusivamente en la lógica sólo reproducen una parte de los conocimientos que una persona inteligente pone en juego para resolver un problema difícil.

En la actualidad existen docenas de programas que tratan problemas técnicos difíciles en áreas tan diversas como el diagnóstico médico, la planificación de experimentos genéticos, la prospección geológica y el diseño de automóviles. La fuente de poder primaria de esos sistemas expertos es su capacidad de razonamiento informal, basada en amplios conocimientos, cuidadosamente obtenidos a partir de expertos humanos. En la mayoría de programas, los conocimientos se codifican en forma de centenares de reglas empíricas "si..., entonces", esto es, reglas heurísticas. Las reglas restringen la búsqueda guiando la actividad del programa hacia las soluciones más probables. Además (y ello distingue los programas guiados heurísticamente de los fundados en métodos más formales), los sistemas expertos son capaces de explicar todas sus inferencias en términos comprensibles para un ser humano. Y pueden hacerlo porque en lugar de basarse en las reglas abstractas de la lógica sus decisiones se valen de reglas dictadas por expertos humanos.

Considérese, por ejemplo, MYCIN, un programa desarrollado por Edward H. Shortliffe, de la Universidad de Stanford, para diagnosticar infecciones bacterianas. Afronta el problema de determinar, entre muchos, el organismo responsable de una infección, para luego, basándose en dicho diagnóstico, recomendar un tratamiento adecuado. Para ello, MYCIN utiliza una base de conocimientos formada por 500 reglas heurísticas, de las cuales la siguiente es un ejemplo típico: "*Si* (1) la tinción del organismo es gram-positiva, (2) la morfología del organismo es la de un coco y (3) la configuración del crecimiento del organismo es en racimos, *entonces* cabe suponer (0,7) que el organismo es un estafilococo". El programa conversa con el usuario, pidiendo información adicional sobre el paciente para aplicar otras reglas, sugiriendo, a veces, la realización de análisis clínicos. En cualquier momento, el usuario puede pedir a MYCIN que le justifique una pregunta o le explique una deducción haciendo explícita la regla que ha aplicado. El programa ha demostrado ser capaz de funcionar a la par con profesionales humanos.

Además del razonamiento heurístico, los sistemas expertos están conectados a otras fuentes de poder, algunas tan próximas al sentido común que raramente la gente tiene conciencia de su empleo. Muchos programas, por ejemplo, pueden enfocar su búsqueda en la medida en que se orientan hacia objetivos más o menos específicos. MYCIN es de ello también un buen ejemplo. A partir de la información general sobre el paciente, MYCIN, cuyo objetivo es hallar la identidad del organismo causante de la enfermedad, razona hacia atrás, planteando preguntas sobre los síntomas específicos que puedan fundamentar el diagnóstico. Una vez determinado, por ejemplo, que "la tinción del organismo es gram-positiva", en el proceso de decidir si las bacterias son estafilococos MYCIN preguntaría inmediatamente sobre la morfología del organismo infeccioso. Esa característica, la de estar dirigido por objetivos, no significa que el programa tenga grabadas sus secuencias de decisiones, como han sugerido, a veces, quienes argumentan que, en realidad, los sistemas expertos no manifiestan inteligencia; los programas como MYCIN se adaptan, de hecho, a situaciones imprevistas por el programador, el cual no predetermina estricta-



3. UN SISTEMA EXPERTO se vale de los conocimientos existentes en una especialidad para restringir la búsqueda de soluciones. DENDRAL, el primer sistema experto y uno de los que más éxito ha alcanzado, se basa en la interacción sinérgica de cuatro tipos de conocimientos para limitar el número de candidatos a estructura de una determinada molécula orgánica. Si sólo se conoce la fórmula, C_{20} H_{43} N, caben 43 millones de configuraciones de los átomos. Conocimientos básicos de topología química, como que el átomo de carbono tiene cuatro enlaces, reducen el número de candidatos a 15 millones. El modelo de fragmentación de una molécula en un espectrómetro de masas, junto a conocimientos heurísticos sobre la estabilidad de las estructuras moleculares y, por tanto, sobre su plausibilidad, limitan aún más la búsqueda. Finalmente, los datos de la resonancia nuclear magnética permiten identificar la estructura correcta.

mente el uso que el programa hará de sus conocimientos.

Otra potente estrategia utilizada por los seres humanos inteligentes, diseñadores de programas de ordenador incluidos, es la de descomponer un problema complejo en subproblemas más manejables: es la estrategia del divide y vencerás. Un grupo de la Universidad de Carnegie Mellon ha construido cuatro programas interrelacionados, todos ellos guiados por reglas heurísticas, cuyo fin es redescubrir famosas leyes de la física y la química. Mediante esos programas, el grupo está avanzando hacia la comprensión y la mecanización de diferentes aspectos de la formación de las teorías científicas; llegado el momento, los investigadores reunirán las diferentes soluciones parciales en un único modelo de todo el proceso.

En un sentido relativamente estrecho, el enfoque del divide y vencerás está implícito en todo programa de ordenador inteligente. Los programas dirigidos por objetivos dividen la búsqueda en subobjetivos más o menos independientes (nudos del árbol de búsqueda). En un nivel superior, los sistemas guiados por principios heurísticos distinguen el propio problema a resolver del meta-problema: la dificultad de decidir en cada momento qué reglas deben ponerse en juego entre los centenares existentes. El meta-problema se resuelve aparte, mediante un proceso, a menudo complejo (algunas veces con heurística propia), consistente en casar el estado actual de búsqueda con las precondiciones que forman el apartado de condiciones "si" en una regla "si..., entonces".

Los métodos formales, aunque no son el motor de la inferencia, pueden también ser útiles para manejar un sistema experto. Algunos sistemas, por ejemplo, se apoyan en procedimientos lógicos o estadísticos para decidir cuándo no es rentable continuar la búsqueda. Además, dado que las reglas heurísticas "si..., entonces" de un sistema experto expresan relaciones no siempre verdaderas, cada regla puede tener asociado un coeficiente de confianza (cuyo valor era de 0,7 en el ejemplo de MYCIN dado anteriormente). Combinando los coeficientes asociados a cada una de las reglas en una secuencia de inferencias, se mide el grado de confianza que merece la conclusión resultante por medio de la ley de Bayes, o con otros procedimientos formales de teoría de probabilidades.

Las reglas de un sistema experto pueden ser sencillas, e incluso a veces no existe, o casi, organización entre ellas. No obstante, el conjunto es capaz de solventar tareas técnicas difíciles con un nivel de competencia de experto. Se trata de una forma de sinergia, pues es mayor el todo que la suma de sus partes. La sinergia es tan común que casi siempre se da por supuesta; sin embargo, casi todos los sistemas expertos la utilizan como fuente de poder.

Uno de los programas inteligentes de mayor éxito ha sido el primer sistema experto desarrollado: DENDRAL, escrito por Edward A. Feigenbaum y sus colegas, en Stanford, a finales de los años sesenta. Junto con su sucesor, GENOA, se usa hoy en los laboratorios de química orgánica de todo el mundo. A partir de los espectros de masas, de los datos de la resonancia magnética nuclear y de otros tipos de información, DENDRAL deduce la estructura de las moléculas orgánicas.

Al igual que MYCIN, DENDRAL es, en esencia, un diagnosticador. Un tipo de sistema experto diferente es aquel que busca descubrir nueva información, o bien redescubrir, a partir de principios básicos, información ya conocida. Ejemplo de este tipo de programas lo tenemos en EURISKO, que el autor ha desarrollado junto con sus alumnos, en Stanford. Después de suministrar a EURISKO una cantidad relativamente pequeña de información básica sobre el tema a tratar, lo hemos soltado por dominios tan diversos como la teoría de conjuntos, juego de guerra, la programación de ordenadores y la limpieza de vertidos químicos.

Durante el proceso de búsqueda, consistente en sintetizar, analizar y evaluar nuevos conceptos, EURISKO se guía por centenares de reglas heurísticas totalmente generales. Entre ellas, tenemos la que conduce a "examinar los casos extremos". Al ponderar EURISKO la función "divisores de", tal planteamiento le condujo a considerar los números con pocos divisores. Así redescubrió los números primos, que son números con sólo dos divisores; también que cualquier número puede factorizarse en un conjunto único de primos. Tan

sencillo planteamiento se demostró inestimable al jugar EURISKO y el autor al juego de la guerra llamado Traveller T.C.S., cuyo objetivo es diseñar una flota de guerra que derrote a la del oponente en batallas cuyo desarrollo viene regido por un gran número de estrictas reglas de juego. Después de considerar las reglas, EURISKO diseñó una flota formada casi enteramente por navíos de ataque pequeños y rápidos, muy parecidos a lanchas, entre ellas un navío tan menudo y rápido que resultaba virtualmente indestructible. Los jugadores humanos de Traveller, burlándose de tal estrategia, formaron flotas más convencionales y equilibradas en el tamaño de los barcos; cayeron ante EURISKO.

Otra regla heurística de gran aplicabilidad es la llamada de "coalescencia", cuyo objetivo es conducir el programa a considerar lo que le sucede a una función de dos variables, *x* e *y*, cuando se les asigna el mismo valor. Tras definir la suma y la multiplicación a partir de la teoría de conjuntos, la regla de coalescencia condujo a EURISKO a descubrir la duplicación (*x* más *x*) y a elevar el cuadrado (*x* por *x*). Aplicando la coalescencia al Traveller, EURISKO desarrolló una nueva estrategia consistente en hundir los navíos puestos fuera de combate, haciéndoles disparar sobre sí mismos. Resultaba un procedimiento razonable para aumentar el poder de la flota, ya que las reglas convencionales del juego definían la agilidad global de la flota a partir de la del navío más lento. Finalmente, al estudiar la programación de ordenadores, EURISKO consideró la función "*x* llama *y*", donde *x* es un componente del programa que activa a otro componente, *y*. La heurística de coalescencia llevó a EURISKO a definir la importante noción de llamada recursiva, o componente de un programa que se llama a sí mismo.

La "coalescencia" y el "examinar los casos extremos" son ejemplos de reglas heurísticas que conducen a un programa "descubridor" a definir nuevos conceptos. Si la misión del programa es descubrir ideas interesantes, debe tener, además, un segundo tipo de criterio heurístico que le ayude a decidir qué conceptos, de los muchos que genere, valen la pena. Las reglas de síntesis de conceptos conducen inicialmente el proceso de búsqueda y, cuando ésta ha comenzado, la regla de evaluación la canaliza por vías fructíferas. EURISKO contiene reglas del tipo siguiente: "si, sorprendentemente, todos los elementos de un conjunto satisfacen una propiedad rara, entonces debe aumentar el interés por el conjunto y por la regla que ha conducido a su definición". Cuando el programa debe decidir qué concepto estudiar entre dos muy semejantes, otra regla le lleva a escoger el concepto que requiere menos tiempo de procesamiento o menos intervención por parte del usuario.

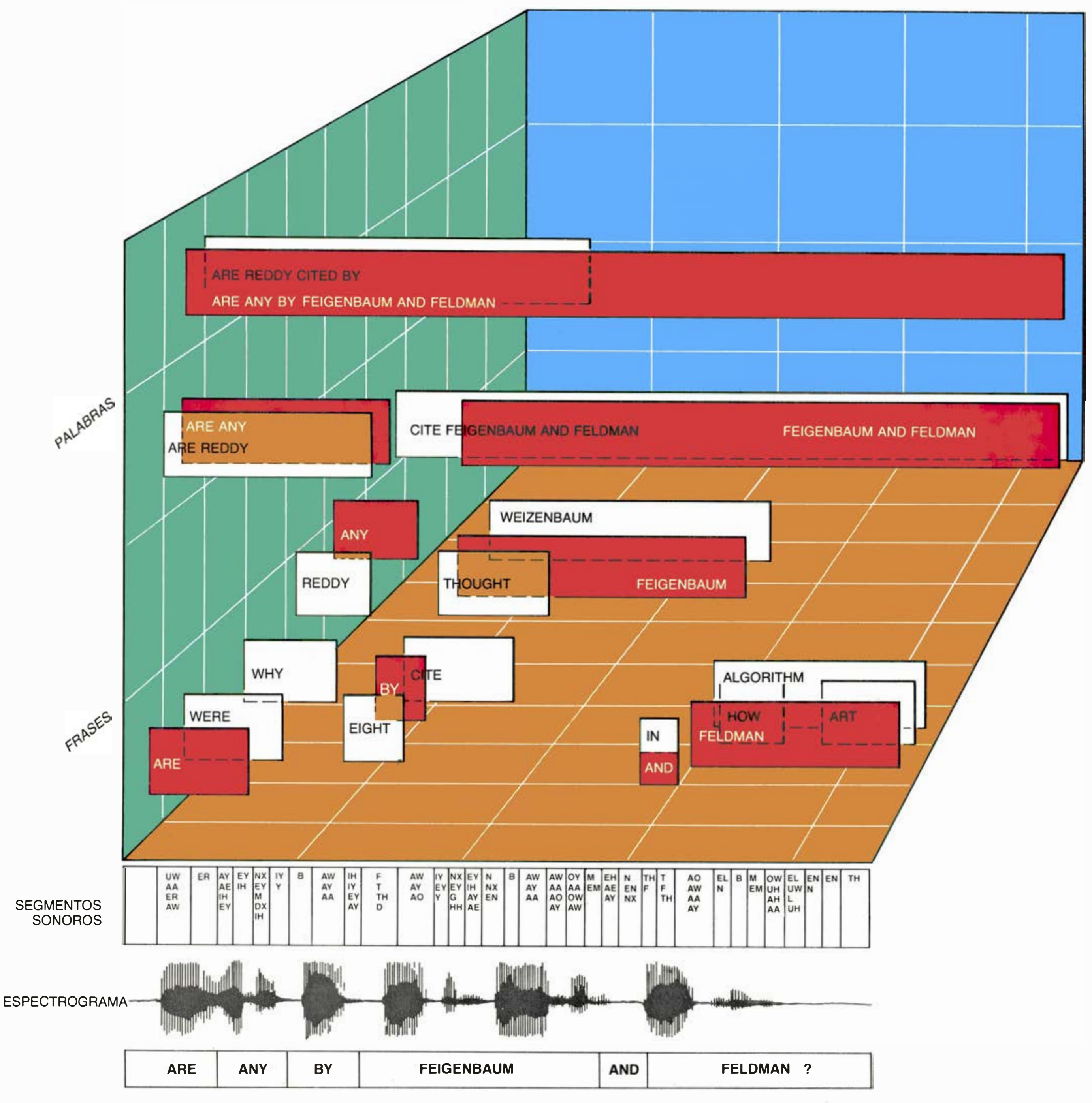
Un corto camino teórico separa el uso de la heurística para descubrir (o redescubrir) nuevos conceptos y hechos, de su uso para descubrir nuevas reglas heurísticas. Este último empeño guarda relación con lo que ha venido siendo objetivo central de la investigación en inteligencia artificial: escribir programas capaces de aprender a partir de su propia experiencia. En los últimos años, cierto número de investigadores han desarrollado programas capaces de formular reglas generales extraídas de su propia experiencia en resolver problemas particulares. Tales procesos de generalización vienen controlados por la llamada meta-heurística.

El éxito de DENDRAL estimuló a sus autores a escribir un nuevo programa, META-DENDRAL, cuyo objetivo era formular reglas generales de espectroscopía de masas basándose en observaciones de la fragmentación de diferentes compuestos en el espectrómetro. Un ejemplo de meta-heurística, en este caso, lo constituye el sencillo enunciado de que las características más importantes de una molécula a la hora de determinar su modelo de fragmentación son aquellas que posee cerca de los puntos de rotura. Aplicando ese criterio heurístico META-DENDRAL pudo formular otro que establece que las moléculas orgánicas tienden a fragmentarse en aquellos puntos donde los átomos de carbono y de oxígeno están unidos por enlaces monovalentes. La nueva regla ayudaría, entonces, al establecimiento de la estructura de moléculas desconocidas a partir de sus espectros de masas. De forma parecida, Thomas M. Mitchell y Paul E. Utgoff, de la Universidad de Rutgers, han escrito un programa llamado LEX2 que deduce reglas heurísticas para el cálculo de integrales a partir de su propia experiencia en la resolución de integrales.

El diseño de programas que puedan aprender con más eficacia depende, en parte, de que se encuentre la manera de conectar con la fuente de poder situada en el centro de la inteligencia humana: la capacidad de comprender y de razonar por analogía. Bastan un poco de introspección y una mirada atenta para advertir que la gente utiliza constantemente la analogía para explicar y

para entender los conceptos, igual que para encontrar otros nuevos. La explotación de esa fuente de poder en los programas inteligentes está en mantillas, pero, sin lugar a dudas, será el foco de la investigación futura.

No quiero decir con ello que no haya habido progreso hasta ahora. Hace 20 años, Thomas G. Evans, del Instituto de Tecnología de Massachusetts, escribió un programa capaz de reconocer analogías entre figuras geométricas, problema éste que forma parte de las pruebas comunes de inteligencia. Lograr que los programas descubran analogías conceptuales ya es más difícil, y en ello trabaja un cierto número de investigadores. Jaime G. Carbonell, de la Carnegie-Mellon, dispone de un programa que reconoce la similitud entre dos algoritmos cuyo objetivo sea el mismo aunque estén escritos en distintos lenguajes de programación, EURISKO, por otra parte, más que encontrar analogías, se vale del razonamiento analógico de bajo nivel. Trabajando en el diseño de circuitos integrados, por ejemplo, EURISKO encontró de improviso que la simetría es una propiedad interesante para esos circuitos, aunque



4. LA PIZARRA constituye un medio de organizar grandes cantidades de conocimientos en un programa inteligente. La información se almacena en módulos independientes, cada uno de los cuales controla sólo una pequeña región de la pizarra y se activa únicamente cuando su región recibe mensajes de otro módulo. Este diseño modular ayuda a resolver el problema de decidir qué porción de la base de conocimientos hay que aplicar en un momento dado. En el ejemplo presentado, adaptación de un sistema de comprensión del habla desarrollado por Raj Reddy, Lee D. Erman y sus colegas, el eje horizontal representa el tiempo, cuyo origen coincide con el inicio de la pronunciación, y el eje vertical representa el nivel de abstracción, desde las ondas sonoras, pasando por segmentos sonoros y palabras aisladas, hasta la oración completa. La tercera dimensión indica el nivel de certidumbre asociado a cada hipótesis presente en la pizarra; de las muchas hipótesis posibles en cada momento y a cada nivel de abstracción, las más plausibles se sitúan en la parte frontal (*fondo rojo*). La pizarra permite la interacción de los módulos de conocimiento situados a diferentes niveles; por ejemplo, cuando el programa ha deducido por la entonación que la oración es una pregunta, esta información se aplica a la formación de hipótesis en el nivel de palabras o frases.

no comprendió el porqué; cuando, más tarde, se le enseñó a diseñar flotas para el juego de Traveller, EURISKO decidió hacerlas simétricas y justificó esta decisión refiriéndola a su experiencia anterior en el diseño de circuitos.

Comparado con las capacidades humanas, todo ello es extraordinariamente exiguo. Las pocas dotes de los programas de ordenador para encontrar y usar analogías son más atribuibles a su escasez de conocimientos que a la falta de capacidad de los programadores para descubrir los algoritmos apropiados. La gente dispone de un inmenso almacén de conceptos donde buscar posibles analogías: quizá tenga memorizados más de un millón de distintos objetos, acciones, emociones, situaciones, etcétera. Ese rico repertorio no lo tienen los programas de ordenador existentes, ni tampoco pueden acumular un amplio conjunto de experiencias a partir de las cuales establecer comparaciones. Los programas cuyo tiempo de funcionamiento entre su puesta en marcha y su detención es grande no suelen guardar en registros adecuados su experiencia de búsqueda, y al detenerse pierden todas, o la mayoría, de las lecciones aprendidas. Incluso EURISKO, que a veces ha funcionado durante semanas y ha podido volver a empezar con la mayoría de sus registros intactos, ha mostrado una vida mental muy escasa, con experiencias ni mucho menos tan variadas como las de un niño.

Por tanto, la prescripción para mejorar la capacidad de razonamiento analógico de un programa es la misma que la necesaria para aumentar las prestaciones generales de los programas inteligentes: expandir la base de conocimientos. En principio, podría almacenarse una enciclopedia entera de forma accesible al ordenador, no en texto, sino en una colección de miles de unidades estructurales con índices múltiples. Los trabajos preliminares hacia dicho objetivo, realizados por unos pocos investigadores, han revelado que es más difícil alcanzar esa meta de lo que parece: la propia comprensión de los artículos de una enciclopedia requiere un amplio cuerpo de conocimientos y sentido común que no poseen todavía los programas de ordenador.

Por una parte, los programas de ordenador habrán de disponer de muchos más conocimientos para razonar de manera eficaz por analogía. Por otra, para adquirir tal volumen de conocimientos parece necesario que los ordenadores comprendan al menos las analogías que se les presenten, pues ésta es, ciertamente, una de las técnicas de aprendizaje más poderosas de que dispone el hombre. Por tanto, el problema es como el del huevo y la gallina. Afortunadamente, es más sencillo que un ordenador, como una persona, comprenda una analogía al presentársela como tal que pretender que la encuentre por sí mismo. Las investigaciones en curso ofrecen razones para esperar que el dilema no resulte intratable.

La clave para conseguir que una máquina entienda una analogía reside en representar convenientemente la información sobre los objetos que han de compararse: por ejemplo, mediante "cuadros" formados por conjuntos de casillas, donde cada una contiene un valor para un determinado atributo de un objeto. Cuando se comunica al ordenador que dos objetos son análogos ("Federico es como un oso"), puede, simplemente, llenar las casillas vacías de un cuadro con valores tomados de las casillas equivalentes del otro. Lo difícil para un programa ignorante reside en decidir qué valores hay que transferir (¿qué atributos de Federico se parecen a los de un oso?) Tales decisiones pueden guiarse por criterios heurísticos. La regla "examinar los casos extremos", por cierto, también es útil aquí. Cuando una analogía resulta adecuada, a menudo es porque las características poco comunes de uno de los objetos comparados son aplicables al otro.

El empleo de cuadros para mecanizar la comprensión de la analogía ilustra un hecho general, a saber, que la representación del conocimiento es, en sí misma, fuente de poder para un sistema inteligente. Un determinado conocimiento puede representarse de muchas maneras en un programa de ordenador; el autor no intenta aquí examinar todas ellas. La cuestión estriba en que cada modo de representación determina que un programa resulte adecuado para realizar ciertas tareas e inadecuado para realizar otras. Establecer analogías, por ejemplo, podría convertirse en una búsqueda larga y pesada si cada atributo de cada objeto se representase en la base de conocimientos por un enunciado independiente de la lógica formal. Escoger la representación adecuada al problema a resolver puede redundar en una reducción del proceso de búsqueda.

El hombre, no obstante, se desenvuelve bien manejando más de una sola elección para cada caso: tenemos capacidad de alternar entre diferentes formas de representación (palabras, símbolos, imágenes) y también podemos abordar un problema desde diferentes perspectivas mientras buscamos su solución. Tal flexibilidad es difícil de emular mediante programas de ordenador. En 1962, Herbert L. Gelernter diseñó un programa que resolvía problemas de geometría plana a nivel de bachillerato; cada problema se representaba a la vez por axiomas y por un diagrama. La representación lógica permitía al programa construir demostraciones formales. Los diagramas, por su parte, sugerían métodos de demostración y permitían al programa probar conjeturas; por ejemplo, reconocía si dos segmentos eran paralelos, si dos ángulos eran iguales o complementarios, etcétera. Aunque una coincidencia de ese tipo podía ser un artefacto de un determinado diagrama, la probabilidad de tal coincidencia era tan pequeña que confería a la técnica de representación múltiple una gran eficacia a la hora de reducir la búsqueda.

Desafortunadamente, esas ideas prescientes de Gelernter sobre la representación múltiple no se han extendido a otros dominios, aunque algunos investigadores han empezado ya a clasificar distintas formas de representación y a trabajar con técnicas que permitan a un programa pasar de una forma a otra. Sin embargo, los diagramas de los programas de Gelernter, además de resultar eficaces por cuanto suponían una nueva técnica de representación, lo eran por su carácter analógico: sus componentes correspondían a entidades reales y las distancias entre componentes se correspondían bien con las distancias reales, como sucede en los mapas de carreteras. Es esa una de las ventajas que las representaciones lógicas no pueden ofrecer, y algunos investigadores exploran la manera de sacar partido de la capacidad, potencialmente grande, de las representaciones analógicas.

De esas líneas de investigación, una merece comentario aparte. Las "pizarras" no constituyen procedimientos para representar componentes individuales del conocimiento, sino una forma de organizar esos componentes en grandes programas; una pizarra representa el propio espacio del problema. En la comprensión del habla, donde primero se aplicó este enfoque, el eje horizontal de la pizarra representa el tiempo, teniendo un enunciado su origen a la izquierda y, su final, a la derecha. El eje vertical mide el grado de abstracción, que crece desde la onda sonora, pasando por las sílabas, hasta

la oración, a la vez que progresa la comprensión de la pronunciación. Cada regla "si..., entonces", o cada conjunto de reglas, dirige una parte de la pizarra y sólo se activa cuando esa parte recibe información; así, la pizarra sirve para solucionar el metaproblema consistente en decidir qué reglas hay que activar en cada momento. Además, todos esos módulos de conocimiento, que, por otra parte, operan independientemente, han de ser siempre reglas "si..., entonces". En definitiva, una pizarra es una forma natural de utilizar la sinergia entre diferentes tipos de conocimiento en un sistema único.

Recientemente empieza a sonar en los círculos de la inteligencia artificial otra fuente potencial de poder de la que se habrán de aprovechar los sistemas inteligentes; se trata del paralelismo. En la actualidad, la mayoría de los ordenadores procesan la información secuencialmente, ejecutando una operación tras otra. Sin embargo, varios grupos de investigadores, incluidos los que trabajan en la llamada "quinta generación" de los japoneses y los implicados en los proyectos norteamericanos sobre el "objetivo estratégico de la informática", están diseñando máquinas que contendrán del orden de un millón de procesadores operando en paralelo. La posibilidad de que la velocidad de procesamiento aumente en un factor de un millón ha llevado a algunos investigadores a predecir mejoras revolucionarias en las prestaciones de los programas de ordenador inteligentes.

Las mejoras serán, indudablemente, significativas. El aumento de la velocidad de procesamiento puede facilitar el logro de soluciones para algunos problemas interesantes, así la comprensión del habla por el ordenador con la misma rapidez con que ésta se produce; también bastaría para que las máquinas vencieran al mejor jugador de ajedrez. Pero, antes de predecir los milagros de la quinta generación, debe recordarse que la mayoría de los problemas difíciles tienen árboles de búsqueda que crecen exponencialmente. Incluso un aumento de la potencia de cálculo en un factor de un millón no cambiaría el hecho de que la mayoría de problemas no pueden resolverse con la fuerza bruta, sino sólo a través de una aplicación juiciosa de conocimientos que permitan limitar la búsqueda.

Hay otra razón, más sutil, para no considerar el procesamiento en paralelo como una panacea; se deduce de las pruebas empíricas obtenidas por el autor y sus colegas. Cuando hicimos que EURISKO simulara la acción de un número progresivamente creciente de procesadores trabajando simultáneamente, en paralelo, en tareas de su agenda, observamos que, a partir de los cuatro primeros procesadores, la velocidad a la que el programa lograba descubrimientos importantes no aumentaba. La razón residía en que, al completar la tarea más prioritaria, EURISKO solía encontrar una nueva tarea más interesante que el resto de su agenda original. En definitiva, en aquellos casos en que una buena heurística permite realizar una búsqueda del tipo "primero lo mejor", el procesamiento en paralelo puede llegar a producir beneficios decrecientes.

Hay una última fuente de poder que vale la pena mencionar, aun corriendo el riesgo de que parezca una ironía: la suerte. Aunque uno no puede contar con la suerte para resolver problemas específicos, ésta inspira a menudo confianza, en un sentido estadístico. Por ejemplo, Woodrow W. Bledsoe, de la Universidad de Texas en Austin, encontró que valía la pena incluir en su programa de demostración de teoremas la siguiente regla heurística de "probar suerte": "Al deducir una nueva proposición, independientemente de si resuelve o no el problema en curso, debe probarse si resuelve alguno de los objetivos de nivel superior".

Hasta cierto punto, todos los científicos experimentales confían en la suerte cuando van acumulando datos con la esperanza de encontrar algún modelo que los explique. Los programas que aprenden empíricamente, como EURISKO, cuya misión es buscar nuevos conceptos y regularidades, dependen igualmente de su buena luna. Esa actividad, que en principio no tiene limitaciones, puede hacerse con menos riesgo si la búsqueda se confina a áreas donde se sabe que existe gran densidad de respuestas interesantes. Sin embargo, la utilización plena de la fortuna exige a los diseñadores una clara voluntad de riesgo al utilizar programas cuyas prestaciones están lejos de la garantía. Aunque las universidades y las empresas suelen actuar de ese modo con los científicos, quizá transcurran muchos años antes de que éstos y los mismos diseñadores de programas dejen de ser reacios a asumir tales riesgos en los programas inteligentes.

Todas las fuentes de poder que he descrito adquieren sentido, y son aplicables, gracias a ciertas propiedades del dominio del problema, y resultan rentables gracias a otras. Ambos tipos de propiedades son, en gran medida, condiciones necesarias y suficientes para aplicar las citadas fuentes de poder. Considérese el caso de la analogía: ésta sólo tiene sentido entre dos conceptos si éstos comparten muchos nombres de atributos, y es rentable si, además, los conceptos son realmente similares en algunas de sus cualidades, esto es, si ciertos valores de sus atributos son comparables. Por ejemplo, la mayoría de enfermedades tienen en común algunos nombres de sus atributos ("causa", "síntomas", "tratamiento", etcétera) y, por tanto, cabe establecer analogías entre ellas. Hacerlo es útil, puesto que las enfermedades cuyas causas son similares requieren frecuentemente tratamientos similares. De hecho, los estudiantes de medicina a menudo aprenden enfermedades nuevas por analogía con otras que ya habían estudiado; algún día los programas informáticos de diagnóstico médico harán lo mismo.

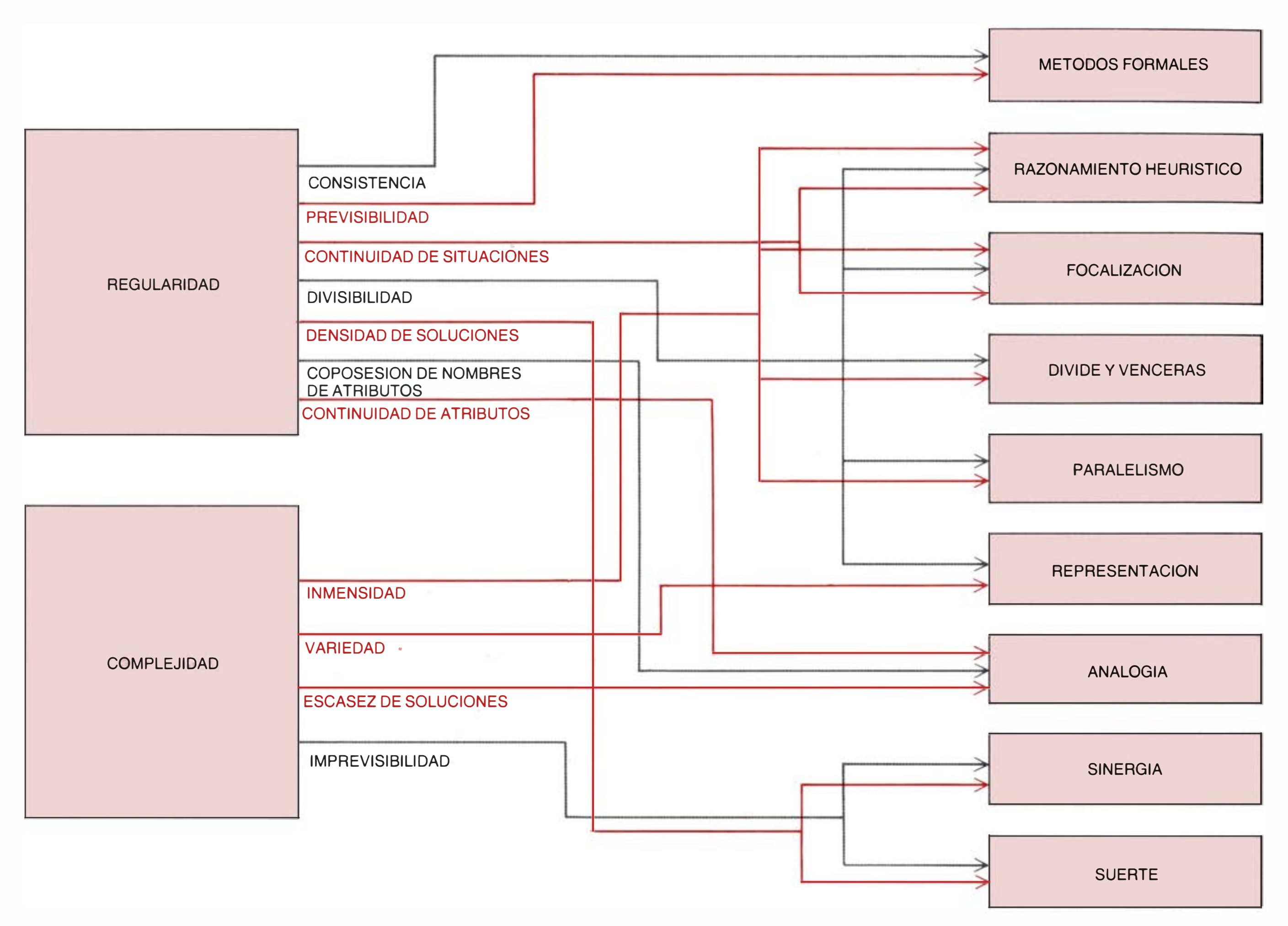
Una característica común a muchos problemas interesantes es la de ser inmensos. Tal inmensidad suele considerarse un obstáculo a superar, pero, al mismo tiempo, hay que contemplarla como una oportunidad para el que re-

"FEDERICO ES COMO UN OSO"

NOMBRE	FEDERICO
ES-UN	HOMBRE
VIVE	CALLE MAYOR, 15
TALLA	GRANDE
ANDARES	PESADOS
COME	MIEL
EDAD	
UÑAS	LARGAS

NOMBRE	OSO-TIPICO
ES-UN	OSO
VIVE	CUEVA
TALLA	GRANDE
ANDARES	PESADOS
COME	MIEL
EDAD	5
GARRAS	LARGAS

5. EN UN CUADRO cabe representar el conocimiento de un determinado concepto; entre otras ventajas, facilita la obtención de analogías por parte de un programa inteligente. Está constituido por casillas, ocupadas por atributos y sus valores asociados. Si dos objetos comparten algunos atributos, puede establecerse una analogía sin más que rellenar las casillas vacías de un cuadro con valores de los atributos apropiados del otro cuadro. Las reglas heurísticas guían el programa en la determinación de los valores a transferir, dirigiéndole, por ejemplo, a considerar las cualidades extremas del cuadro modelo. Si una casilla del modelo no se corresponde con otra del cuadro destinatario, el programa escoge una similar.



6. FUENTES DE PODER para la resolución de problemas. Adquieren sentido (*líneas grises*) y son realmente útiles o eficaces (*líneas de color*) de acuerdo con ciertas propiedades de la temática del problema. Por ejemplo, cabe aplicar razonamientos heurísticos, o el procedimiento "divide y vencerás", a un problema si éste es regular en el sentido de que es descomponible en subproblemas. Sin embargo, su aplicación es provechosa sólo si el dominio del problema es complejo e inmenso; si es limitado y regular, quizá convenga mejor aplicar un enfoque de la lógica formal. Similarmente, las analogías pueden obtenerse más fácilmente en un dominio (como el diagnóstico médico) en que los objetos (enfermedades) tienen en común muchos nombres de atributos. Razonar por analogía sólo es provechoso cuando existe una continuidad en los valores de los atributos (enfermedades con síntomas y causas semejantes requieren, a menudo, tratamientos semejantes) o cuando se tiene pocas soluciones (un conjunto de síntomas está ligado a unas pocas enfermedades).

suelve problemas. Si el espacio de búsqueda es grande, podemos intentar resumir algunas de sus partes mediante estadísticas, teoremas o reglas heurísticas. La oportunidad de hacerlo escapa en el caso de problemas cuyo espacio de búsqueda no sea inmenso, aunque presenten gran dificultad debido al tiempo de resolución que consumen; un buen ejemplo de eso último lo constituye el control de los efectos secundarios a largo plazo de buen número de medicamentos.

Cuando los seres humanos se enfrentan a un problema complejo, ponen intuitivamente en juego todas las fuentes de poder apropiadas al problema. Los primeros programas de inteligencia artificial, en cambio, tenían la grave debilidad de fiar en una única fuente de poder, normalmente algún método formal. Muchos diseñadores de programas reconocen actualmente la importancia de aprovechar toda la gama de técnicas humanas para la resolución de problemas, así como la importancia de la sinergia que resulta de la cooperación concertada entre diferentes fuentes de poder.

Quizá con la excepción de la sinergia y de la fortuna, las fuentes de poder tratadas son todas métodos de organización y aplicación de los conocimientos orientados a reducir el proceso de búsqueda. Si el futuro de la inteligencia artificial pasa por dotar a las máquinas de esos medios humanos, entonces quizás el futuro dependa meramente de la capacidad de los programadores para aportar a sus sistemas la materia prima adecuada: la enorme base de conocimientos sobre hechos y experiencias que fundamentan el razonamiento humano. Hasta cierto punto, tales conocimientos podrían incorporarse "manualmente" al sistema merced a un programador que realizara todo el trabajo. Sin embargo, la emulación por las máquinas de muchos de los más impresionantes logros intelectuales del hombre será imposible mientras los programas no se parezcan más a los seres humanos en dos aspectos fundamentales: primero, en su capacidad para acumular sus propias experiencias durante un período largo de vida mental y, segundo, en su capacidad para comunicarse y para aprender unos de otros.

Diseñar una programación de ordenadores que se ajuste a esta descripción es un desafío tremendo, pero estoy convencido de que algún día se logrará. La mayoría de los programas existentes fueron diseñados pensando en un entorno estático. En aquellas áreas donde los conocimientos cambian rápidamente, y con ellos también los problemas (arquitectura de ordenadores, diseño de circuitos integrados y biotecnología), esa característica del diseño se nos aparece como un serio obstáculo; los programas que operan en esas áreas caducan rápidamente. La capacidad de adaptación a los cambios del medio exige inteligencia. Así, entiende el autor que la inteligencia de los programas se demostrará cada vez más una necesidad y no un lujo.

Juegos de ordenador

La flaca vista de un ojo digital hace pensar que no puede haber visión sin comprensión

A. K. Dewdney

Imaginemos una caja negra de apariencia similar a una cámara fotográfica. En su cara frontal, un objetivo; en un costado, un selector con diversas posiciones: "árbol", "casa", "gato", etcétera. Con el selector en la posición "gato" salimos a dar un paseo. Al poco, sentado en el zaguán de un vecino descubrimos un gato. Al orientar la cámara en su dirección se enciende una luz piloto roja. Enfocando cualquier otra cosa, la luz sigue apagada.

En el interior de la caja se ha instalado una retina electrónica que envía impulsos a una red lógica biestratificada. He aquí un ejemplo del dispositivo llamado perceptrón. Se tuvo en un tiempo la ilusión de que llegaría verdaderamente a disponerse de perceptrones que llevaran a cabo tareas de reconocimiento e identificación de seres del mundo real, como en la fantasía recién descrita. Sin embargo, algo no ha funcionado.

Los decenios de 1950 y 1960 fueron de tremenda inventiva e investigación en el recién roturado campo de las ciencias de cómputo. Muchos científicos sufrieron la influencia de románticos paradigmas, como la posibilidad de máquinas capaces de auto-organización, o de aprender, o provistas de inteligencia. Siento la tentación de llamar a ese período "Edad cibernética". A la vuelta de la esquina parecían encontrarse máquinas increíbles, capaces de ver, de pensar, incluso reproducirse. La más sencilla de tales máquinas era el perceptrón.

Un perceptrón consta de una retina finita, matricial, semejante a una fina malla de celdillas receptoras de luz. Lo mismo que ciertas células de la retina humana, las celdillas del perceptrón se excitan si reciben suficiente luz; de lo contrario permanecen inertes. Resulta, pues, razonable considerar la imagen que el perceptrón analiza como un cuadriculado de casillas oscuras y claras, como se muestra en la figura 1.

Además de la retina, el perceptrón dispone de una multitud de elementos de decisión primaria, que llamaré diablillos locales. Cada diablillo tiene a su cargo el examen continuo de un subconjunto fijo de la matriz de células retinianas, y debe informar de las condiciones de su zona a un elemento decisorio más complejo, que bien podríamos llamar un demonio-jefe. Concretamente, cada diablillo local está provisto de un catálogo de pautas, a cuya aparición debe estar atento, por si se dieran en su demarcación, que es el subconjunto de células retinianas a su cargo. De darse alguna de las pautas catalogadas, el diablo local enviará una señal al demonio jefe; en los demás casos permanece en silencio. El trabajo encomendado al demonio jefe es algo más complicado, porque tiene que efectuar ciertos cálculos aritméticos. Las señales que cada uno de los diablos locales envían se multiplican por un entero positivo o negativo (el "peso" atribuido a cada diablo local), y se suman los productos resultantes. Si el valor de la suma es mayor o igual que cierto valor umbral fijo, el demonio jefe pronuncia un "sí"; en caso contrario dice "no". Para no fantasear sobre el aspecto de los distintos demonios, en la figura 1 los he representado por cubos.

Es cosa frecuente asignar a los diablos tareas peligrosas o imposibles, como las de abrir y cerrar minúsculos portillos en las paredes de un recipiente para dejar pasar moléculas. Comparados con éstos, los diablillos del perceptrón tienen tareas muy fáciles. Tanto así, que los demonios locales pudieran remplazarse por circuitos lógicos sencillos y, el trabajo del demonio jefe, por unos cuantos registros, un sumador, y un comparador (elementos de la unidad central del procesador de cualquier ordenador). Los demonios, empero, tienen un encanto romántico inalcanzable por los dispositivos electrónicos.

La tarea de un perceptrón consiste en decir sí cuando se le presenten ciertas pautas; y no, a todas las restantes. Se dirá que el perceptrón reconoce, o identifica, las primeras. Aunque es dudoso que pueda llegar a construirse un perceptrón capaz de identificar gatos, otro tipo de tareas de reconocimiento sí son verosímiles.

Ajustando la ponderación y el valor umbral, el perceptrón puede programarse, en cierta medida, para reconocer una clase especial de motivos o patrones. Se asignan pesos positivos a los diablillos locales que proporcionen pruebas o indicaciones de pertenencia a esa clase y, pesos negativos, a los que aporten indicaciones en contra. La magnitud del peso refleja la importancia conferida a la indicación. Aunque los perceptrones que aquí comentamos operan con sistemas fijos de pesos, la noción de programación desempeña papel fundamental en la teoría de perceptrones, desarrollada a mediados del decenio de 1950.

El perceptrón que ahora explicaremos es capaz de reconocer un rectángulo oscuro, de tamaño cualquiera, situado en cualquier región de su retina. Más aún, reconoce cualquier número de tales rectángulos (incluida su ausencia) con tal de que ningún par de ellos se toque a lo largo de un lado, o en un vértice. La construcción del perceptrón comprende tres fases. Primero, hay que instalar un demonio local en cada submatriz de 2×2 de la retina. Después, se ponen en la lista-catálogo de cada uno de los demonios locales todas las subpautas de una lista P [*véase la figura 2*]. Tercero, se ajustan al valor +1, en el demonio-jefe, los pesos asignados a todos los demonios locales, y si su número es d, se ajusta a d el valor umbral.

Tal diseño requiere una buena cohorte demoníaca. Si la retina del perceptrón consta de $n \times n$ celdillas, tendrá que haber $(n - 1)^2$ demonios locales. Todos tienen asignado peso positivo, esto es, todos proporcionan pruebas de carácter afirmativo sobre la identificación de rectángulos. No es difícil ver, por ejemplo, que cuando la imagen proyectada sobre la retina de un perceptrón es un rectángulo, toda submatriz cuadrada de 2×2 celdillas ha de contener alguna de las subpautas de la lista P. Se sigue que cada demonio local envía una señal al demonio-jefe, y que la suma ponderada de todas las señales es, evidentemente, d. El demonio jefe se pronuncia afirmativamente. Por otra parte, si alguna de las formas oscuras no fuese un rectángulo, o si dos rectángulos se tocasen, al menos uno de los dominios locales de 2×2 contendría una subpauta incluida en la lista N de la figura 2. Por consiguiente, al menos uno de los demonios deja de informar, y el demonio-jefe obtiene una

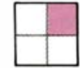
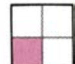
suma no mayor que $d - 1$. Su respuesta es negativa.

Podría construirse un perceptrón equivalente en el que cada demonio local utilizase la lista N, más breve. En este caso, todos los pesos habrían de ser -1, y el umbral se situaría en 0. Cada diablo local proporcionaría pruebas de carácter negativo sobre existencia de pautas de rectángulos, y el demonio jefe solamente diría que sí cuando ninguno de los demonios locales enviase una señal.

El estilo de perceptrón recién definido tiene interesantes propiedades, y vale la pena darle nombre. Sin especificar qué lista de subpautas utilizarán los diablos locales, los dispositivos de esta clase serán todos llamados perceptrones de ventana, porque cualquiera de los demonios locales observa el motivo que se le proporciona al perceptrón a través de una ventana de 2×2. Para retinas de $n \times n$ habrá necesariamente $(n - 1)^2$ diablos locales; el valor umbral será igual a este número.

Hablando a grandes rasgos, la tarea que mejor realizan los perceptrones es el reconocimiento de figuras geométricas. Los perceptrones de ventana no sólo reconocen rectángulos, también "agujeros negros", esto es, casillas oscuras aisladas, líneas rectas verticales y horizontales, escaleras, escaqueados y muchos otros motivos. Todo depende del sistema de subpautas de tamaño 2×2 que se elija como lista-catálogo de los demonios locales [*véase la figura 4*]. Más todavía, cada subconjunto de las 16 posibles subpautas de 2×2 define un perceptrón diferente, y cada uno de los 65.536 perceptrones de ventana resultantes reconoce una clase específica de motivos.

¿De verdad? El perceptrón de ventana basado en las dos subpautas mostradas a continuación no reconoce nada:

La razón es sencilla. Supongamos una retina francamente grande, y seleccionemos una ventana de 2×2 en algún lugar cercano al centro. Si la ventana contiene la primera de las subpautas anteriores, examinemos la ventana situada una casilla hacia la derecha: tendrá una casilla oscura en su ángulo superior izquierdo, y, por consiguiente, el diablo encargado de esa submatriz no enviará al demonio-jefe ninguna señal. Recordemos que en un perceptrón de ventanas *todos* los demonios locales han de informar positivamente para que se reconozca una pauta. Si es la segunda subpauta la que aparece, al desplazar la ventana una casilla hacia la izquierda se obtiene una contradicción similar.

¿Qué subconjuntos de las 16 subpautas dan perceptrones de ventana que realmente reconocen algo? Será ésta cuestión difícil de contestar; pero ilustra muy bien el tipo de preguntas que el matemático o informático interesados pudieran plantearse al verse enfrentados al fenómeno de un perceptrón que no reconoce cosa alguna. Considerado el gran número de tales perceptrones, lo deseable sería disponer de algún criterio de aplicación sencilla: dado un subconjunto de subpautas 2×2, se aplica el criterio y se obtiene la respuesta para ese subconjunto particular.

La intención de esos comentarios es mostrar que no siempre se precisa reputación profesional como informático teórico para responder a tales cuestiones. Aunque van un punto más allá de los problemas que típicamente suelen proponerse en los artículos de recreación matemática, sí exigen el mismo tipo de mentalidad. Los lectores que hayan resuelto al menos uno de los problemas que Martin Gardner proponía en su sección de "Juegos Matemáticos"

1. Un perceptrón se esfuerza en reconocer un gato

de INVESTIGACIÓN Y CIENCIA harán sin duda progresos importantes en la cuestión que se acaba de plantear. En la investigación teórica, lo mismo que en las ciencias experimentales, una solución parcial es preferible a carecer de toda solución.

El trabajo en perceptrones lo inició Frank Rosenblatt, de la Universidad de Cornell, en el decenio de 1950. Rosenblatt y sus colaboradores, tanto de Cornell como de otros centros, llegaron a confiar en las posibilidades de los perceptrones para reconocer formas y configuraciones. El llamado "teorema de convergencia" les aseguraba que, en principio, los perceptrones podrían aprender a reconocer pautas o motivos haciendo que los pesos asignados por el demonio-jefe se ajustaran a un control automático. El teorema afirma que cualquier reajuste de pesos que mejore las facultades de reconocimiento sirve de base, a su vez, para mejoras ulteriores. Se construyeron perceptrones reales y en ciertas pruebas efectuadas sobre pautas sencillas se alcanzaron altos índices de reconocimientos acertados.

Lo que en aquella época parecían estimulantes progresos resultaron, hasta cierto punto, ilusiones. Bajo la superficie se ocultaban varios defectos graves de concepto. Según Marvin L. Minsky y Seymour Papert, del Instituto de Tecnología de Massachusetts, los entusiastas investigadores del perceptrón habían sido seducidos por la sencillez de sus ingenios y el aparente éxito inicial. En 1969, Minsky y Papert publicaron el libro *Perceptrons*, que pinchó eficazmente el globo, poniendo de manifiesto (y demostrando) algunas de las cosas que los perceptrones no pueden hacer.

Uno de los fallos más serios que Minsky y Papert descubrieron fue la incapacidad de ciertos perceptrones para descubrir si una figura era conexa (es decir, si constaba de una sola pieza). Suponiendo que cada diablillo tenga a su cargo la inspección de una zona limitada, Minsky y Papert dieron ejemplos de cuatro configuraciones, ideadas de modo que todo perceptrón concebido con el propósito de reconocer la propiedad de conexión se confundiera ante, al menos, una de las cuatro. Se muestran esas configuraciones en la figura 3. Dos de ellas (*b* y *c*) son figuras conexas; otras dos (*a* y *d*), no lo son.

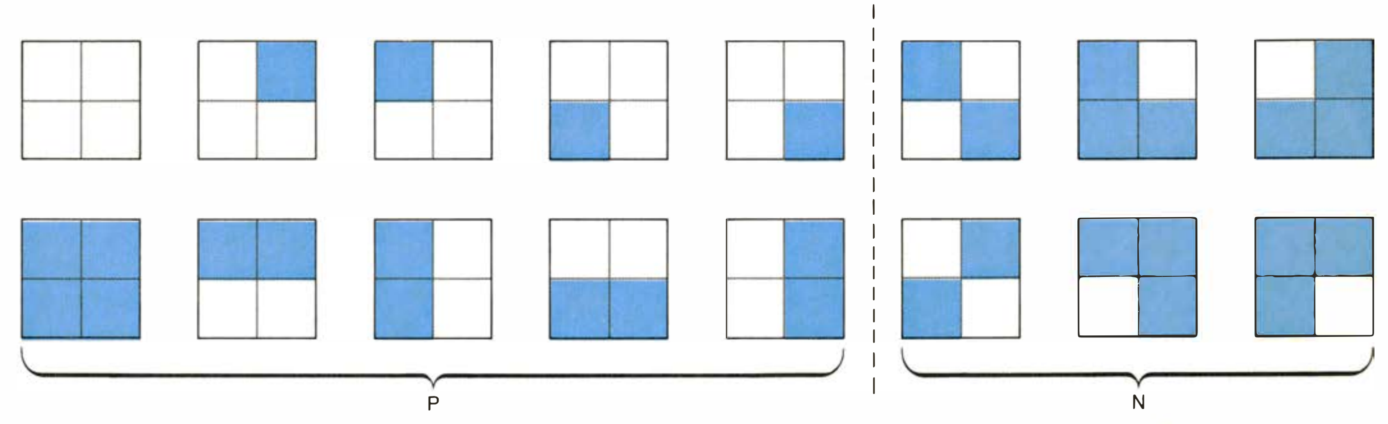
Imaginemos que alguien afirme haber diseñado un perceptrón de "diámetro acotado", capaz de distinguir las configuraciones conexas de las que no lo sean. Con "diámetro acotado" se quiere decir que existe un número m tal que la región examinada por cada demonio local esté siempre contenida en una ventana de $m \times m$. Para poner a prueba tal afirmación, Minsky y Papert prepararían versiones de sus figuras, ajustando la longitud de la configuración para que tuviera más de m casillas de larga. Cabe entonces clasificar los demonios locales en tres conjuntos disjuntos. Los Demonios Izquierdos examinan al menos una casilla situada en el borde izquierdo de la figura. Los Demonios Derechos inspeccionan al menos una casilla del borde derecho. Los demonios que no sean ni izquierdos ni derechos forman la categoría de los Otros Demonios.

Cuando al perceptrón supuestamente capaz de reconocer la conexión se le presenta la configuración *a*, o bien falla (declarándola conexa) o bien acierta (detectando que es inconexa). Evidentemente, si falla, el examen del perceptrón ha concluido. Si acierta, el paso siguiente es examinar la suma efectuada por el demonio jefe, escindiéndola en tres términos: I, D y O, que representan las sumas ponderadas correspondientes a Demonios Izquierdos, Derechos y Otros, que envían señales al demonio jefe al proyectarse en la retina la configuración *a*. Dado que el perceptrón ha declarado la no-conexión de la figura, la suma de I más D más O debe estar por debajo del valor umbral. Sustituyendo ahora la figura *a* por la *b*, sólo los Demonios Izquierdos cambiarán su respuesta, dado que solamente cambian las casillas situadas a lo largo del borde izquierdo de la figura. Sea I' el valor de la nueva suma parcial izquierda. Por otra parte, cambiando la configuración *a* por la *c*, sólo cambiarán las casillas del borde derecho, y nada más cambian su respuesta los Demonios Diestros, pasando de la suma D a la D'.

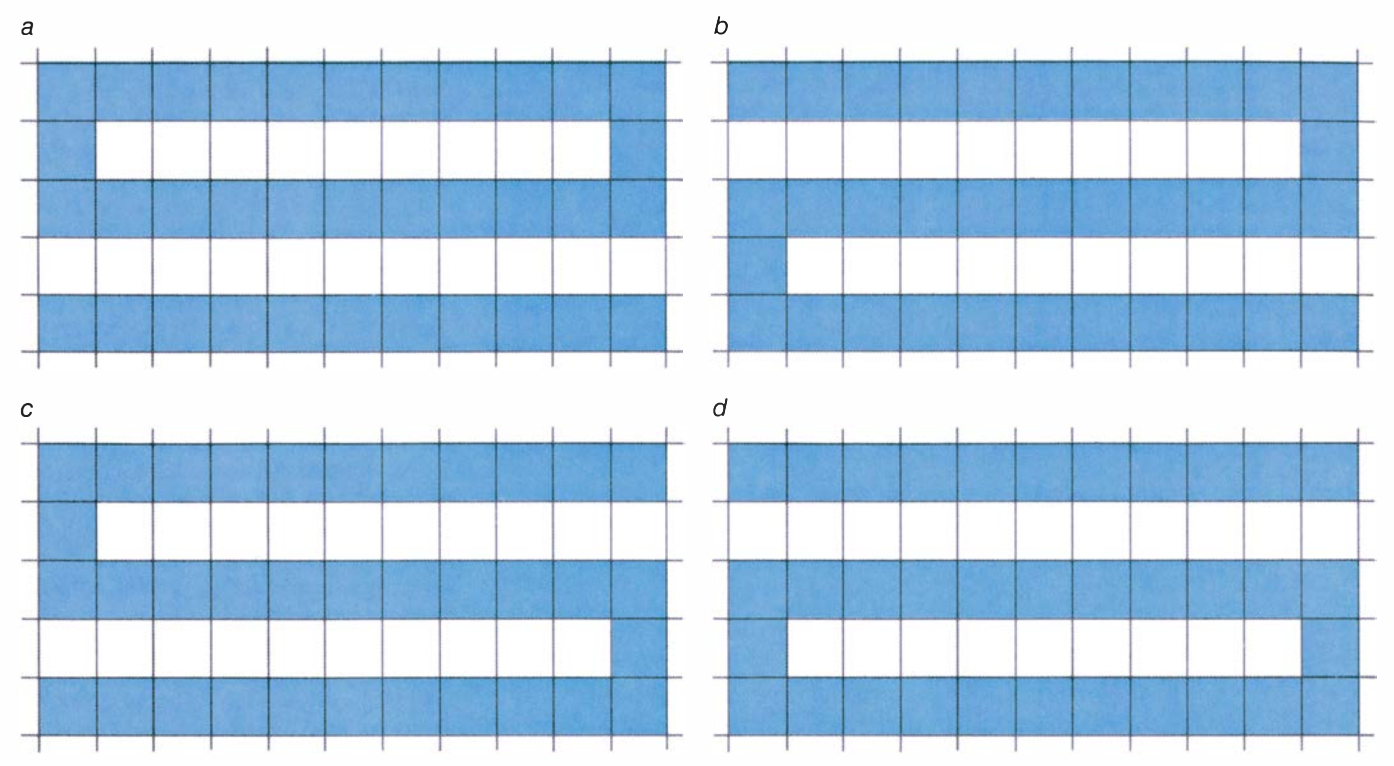
El perceptrón se encuentra ahora en situación harto curiosa. Dado que *b* y *c* son ambas figuras conexas, tiene que declararlo así en ambos casos, y por tanto, las sumas $I' + D + O$ e $I + D' + O$ tienen que ser al menos tan grandes como el valor umbral. Se sabe ya que $I + D + O$ es menor que el umbral, porque *a* no es conexa. Se deduce que I' es mayor que I, y que D' es mayor que D. El golpe mortal llega cuando el perceptrón ha de enfrentarse con la configuración *d*. En este caso cambian con respecto a la configuración *a* tanto las casillas derechas como las izquierdas, y el demonio jefe se encuentra con la tarea de calcular la suma de $I' + D' + O$, que es ciertamente mayor que el umbral: el demonio-jefe declara que la configuración *d* es conexa. Y se equivoca.

Entre otros puntos flacos descubiertos por Minsky y Papert en los perceptrones se cuenta el número inalcanzablemente alto de diablillos locales que exigirían ciertas tareas y la lentitud del ritmo de aprendizaje (o convergencia) de otras.

Tal vez no sea sorprendente que los perceptrones fracasen en muchos casos que el sistema visual humano supera con éxito. Ya hice notar al principio que los diablillos locales y el demonio jefe podían sustituirse por circuitos computacionales sencillos. También podrían sustituirse por neuronas formales, que en el decenio de 1940 describieran por vez primera Warren S.



*2. Subpautas 2 × 2 identificadas por los demonios locales positivos (**P**) y negativos (**N**)*



3. Cuatro figuras ideadas para confundir a perceptrones detectores de conexión

McCulloch y Walter H. Pitts, en su clásica obra sobre redes neuronales. Las neuronas formales son mucho más sencillas que las neuronas humanas; análogamente, la complejidad de un perceptrón, organizado en red neuronal de dos niveles, no se acerca, ni de lejos, a la complejidad de los dos primeros estratos de la corteza visual humana. Además, por así decirlo, "detrás" de la corteza visual se encuentra un aparato analítico admirable y casi totalmente desconocido, algo del todo inexistente en el modelo de visión del perceptrón. Para comenzar siquiera a remedar esta superior complejidad en el modelo perceptrón habría que remplazar al demonio jefe por una máquina de Turing; pero a partir de aquí el razonamiento se hunde en un mar de especulaciones poco fundadas, y por tanto abandonaré en este punto la cuestión.

Aunque los perceptrones sean ojos carentes de mente, son de una simplicidad deliciosa, y al menos para ciertas clases de configuraciones muestran capacidad de reconocimiento bien definida. Me pregunto si los lectores podrían descubrir qué otras configuraciones caen en la esfera de competencia de los perceptrones de ventana. Quienes deseen explorar la cuestión (bastante más difícil) de cuáles son los subconjuntos de las 16 subpautas de 2 × 2 que dan lugar a "buenos" perceptrones de ventana (entendiendo por tales los que reconocen al menos una configuración) encontrarán un poco más clara la cuestión si se añade una restricción más, a saber, ser "trasladables", poder desplazarlas por la retina sin alterar el hecho de ser reconocibles. Esta restricción no sólo descarta ciertos perceptrones superespecializados (por ejemplo, el que reconoce una única casilla oscura situada en el ángulo superior derecho de su retina), sino que refleja también la idea de que un perceptrón mira escenas reales, que pueden desplazarse por la retina conforme la caja negra de mi pequeña fantasía los va explorando.

Si bien los perceptrones de diámetro acotado son incapaces de distinguir las figuras conexas de las inconexas, sí cabe reconocer la propiedad de conexión en ciertas categorías de figuras. Por ejemplo, en la clase de todas las configuraciones multirrectángulos (sin lados ni vértices comunes) las figuras conexas serían las formadas solamente por un rectángulo. ¿Sabría el lector diseñar un perceptrón que reconozca precisamente tales configuraciones? Los diablos locales han de utilizar ventanas de 2 × 2, pero nada impide contratar demonios ayudantes, si son necesarios.

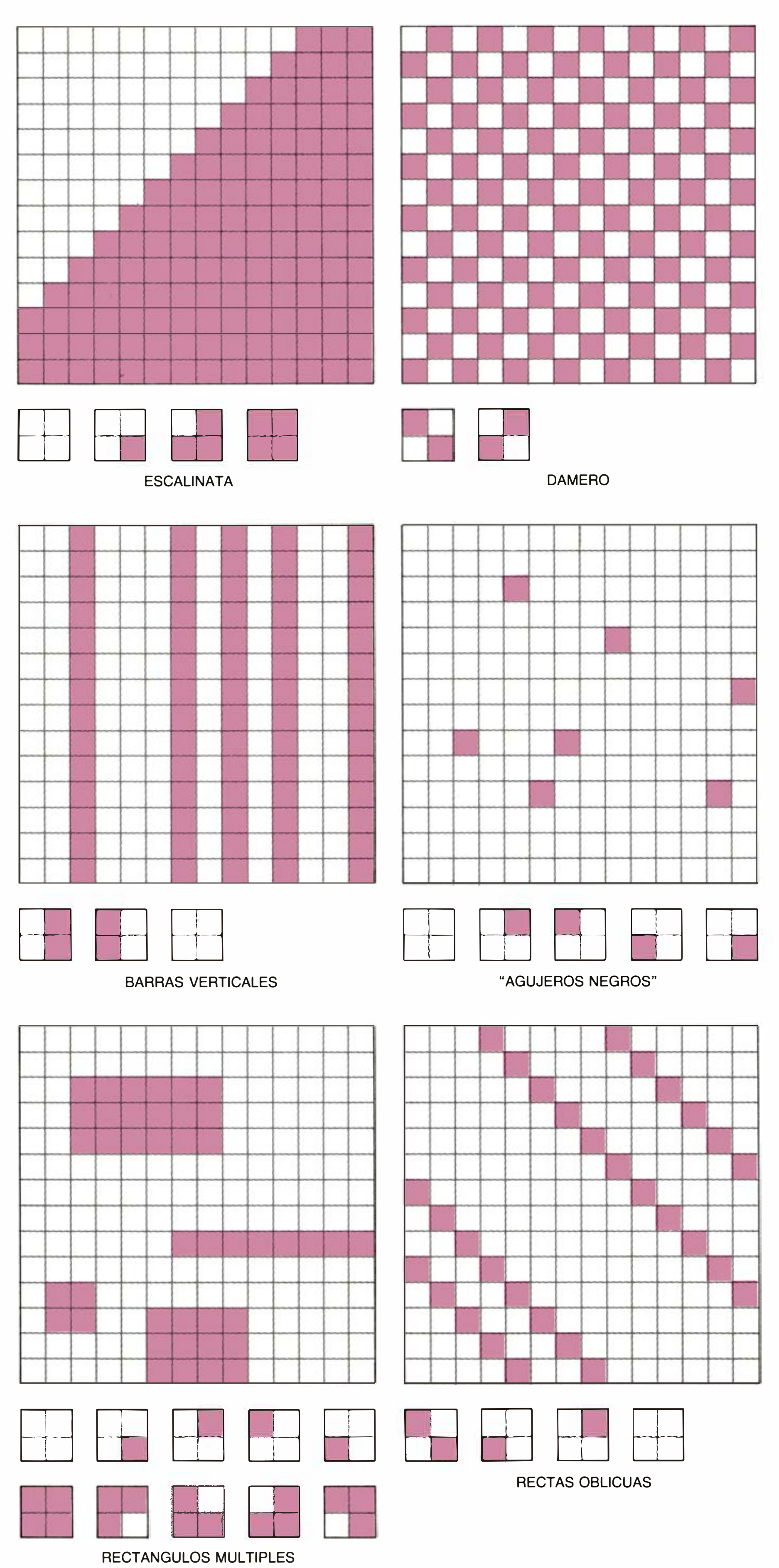
He dado a entender implícitamente que la investigación sobre perceptrones se detuvo tras la publicación de *Perceptrons* por Minsky y Papert. Ello es cierto, en el sentido de que ya no es posible mantener actitudes ambiguas o desiderativas con respecto a los perceptrones o a sus capacidades recognitivas. Por otra parte, nada más lejos de las intenciones de Minsky y Papert que suspender la investigación en teoría de perceptrones. Las facultades de que exactamente puedan disponer esos ingenios, sencillos y, en ocasiones, eficaces, están todavía por descubrir.

Rosenblatt, cuyos trabajos se extenderían hasta campos como la psicología y la neurobiología, murió en un trágico accidente de navegación deportiva, el 11 de julio de 1971, en Maryland.

Muy gratificante ha sido la respuesta de los lectores al artículo sobre artilugios analógicos; se han propuesto nada menos que 17 nuevos artilugios. También he recibido tres soluciones correctas al problema de reflexión de la luz en una caja de paredes azogadas.

No obstante, antes de ocuparme de estas materias debo corregir un error. El algoritmo más veloz del que tengo noticia para calcular la envolvente convexa de un conjunto de n puntos del plano requiere del orden de $n \log n$ operaciones, y no $n \log \log n$. El aparato analógico de cordel que resuelve ese mismo problema lo inventó, en 1957, George J. Minty, Jr., de la Universidad de Indiana. Minty señala también que la técnica de películas jabonosas, utilizada para hallar árboles de Steiner mínimos, la ideó William Wiehle, en 1958.

David Zimmerman, de Beaver Dam, Wisconsin, critica el artilugio láser para descubrir si un número n es primo. La luz tiene que reflejarse n veces para ir desde el láser hasta el detector, hace notar Zimmerman, y dado que la velocidad de la luz es finita, el tiempo exigido por la solución es proporcional a n.



4. Algunos de los motivos que detectan los perceptrones de ventana

Si el tamaño del problema está definido por el número de cifras de *n*, el tiempo de solución crece exponencialmente, y el dispositivo deja de ser intrínsecamente más rápido que los algoritmos digitales.

La finitud de la velocidad de la luz provocó también la disconformidad de Steven P. Hendrix, de New Braunfels, Texas, quien hizo notar que en el problema de la caja de paredes azogadas habría que esperar demasiado a que emergiera la luz. Preguntaba yo qué propiedad de la caja medía la trayectoria de la luz. Hendrix se encuentra entre quienes hicieron notar que la cuestión de si la luz emerge o no de la caja es equivalente a la de si una recta ilimitada del plano intercepta o no algún punto que tenga sus dos coordenadas enteras.

Imaginemos un huerto de frutales, infinito, plantado con una infinidad de árboles infinitamente finos, situados en todas las intersecciones de una retícula cuadrada. Al disparar desde un árbol una bala en una dirección cualquiera, ¿llegará a hacer impacto en otro árbol? Solamente lo hará si la pendiente del ángulo que forma la trayectoria con las hileras de árboles es un número racional, pues si el árbol que recibe el impacto se encuentra *p* hileras hacia el norte y *q* hileras hacia el este del punto de disparo, la pendiente de la trayectoria es p/q. Los espejos de la caja no hacen más que replegar la trayectoria de la bala. También John Dewey Jones, de Farmington Hills, Michigan, y Paul Kingsberg, de Imperial, Pennsylvania, resolvieron el problema.

La única forma de hacer justicia a la abundancia de artilugios descritos por los lectores es dedicar a la cuestión un segundo artículo, presumiblemente, hacia la primavera de 1985. Mientras, mencionaré algunos de los más interesantes.

Peter F. Ash, de la Universidad San José, da cuenta de cómo resolver una ecuación cúbica por inmersión de cuerpos sólidos en un tanque de agua. Tom Digby, de Los Angeles, indica que la capacidad de cómputo de las máquinas analógicas puede atribuirse a la posibilidad que tienen de realizar muchos procesos en paralelo. Demuestra cómo organizar *n* ordenadores digitales para ordenar *n* números en tiempo lineal, igualando así el rendimiento del ordenador analógico de spaghetti.

Eric Halsey, de la Universidad de Washington, describe un artilugio para hallar el trayecto de longitud máxima entre dos puntos de una red, mediante una serie de "serpientes". Cada

lado del grafo está representado por un hilo elástico enfilado a través de un número finito de anillas. Al estirar y luego aflojar el artilugio, ¿no quedará de manifiesto el camino más largo? Otro de sus artefactos mide la longitud del camino más breve que conecta dos vértices de un grafo. Cada lado se materializa con un trozo de mecha para explosivos, cortado a la longitud equivalente; en el segundo vértice se pone un petardo. Ahora damos lumbre a las mechas en el primer vértice, retirándonos prudentemente. El tiempo que tarde el petardo en detonar será proporcional a la longitud del camino más corto.

Palmer O. Hansen, Jr., de Largo, Florida, me recordó que el planímetro, un artilugio mecánico para medir superficies, podía clasificarse entre los artefactos analógicos. Dale T. Hoffman, del Bellevue Community College, de Washington, señala algunos otros problemas resolubles mediante películas jabonosas, entre ellos un perspicaz cálculo de la ley de Snell. David Kimball, de San Diego, resuelve laberintos inyectándoles agua, y siguiendo su correr hasta la salida. Otro precioso artilugio describe J. H. Lueth, de Carteret, Nueva Jersey. Su artilugio determina qué emplazamiento dar a una fundición para hacer mínimos los gastos de transporte de carbón, fundentes o arenas de moldeo y mineral. Tres orificios en una tabla y tres pesos anudados conjuntamente con hilo fijo lo resuelven.

Tony Mansfield, del Laboratorio Nacional Británico de Física de Teddington, resuelve problemas de programación lineal con una estructura montada con elementos de un juguete tipo "mecano". Thomas A. Reisner, de la Universidad Laval de Quebec, genera un mapa topográfico de una superficie extendiendo una gasa fina sobre ella. Una fuerte luz cenital crea efectos "moiré", al interferir la red con su propia sombra.

Al parecer, la industria de agrios se vale también de un dispositivo analógico para clasificar los frutos por tamaño. Se hacen rodar las naranjas entre dos tubos no del todo paralelos; cuando el diámetro del fruto es igual a la separación de los tubos, cae. John P. Schwenker, de Louisville, Colorado, determinó en cierta ocasión el centro de gravedad de una pieza por medio de una variante del método de la bandeja equilibrada, de Ronald L. Graham. Al arrastrar la pieza sobre una superficie horizontal bien lisa, el plano vertical que pasa por la cuerda también pasa por el centro de gravedad. Por intersección de tres de estos planos queda determinado el centro de gravedad.

Taller y laboratorio

Conociendo la mecánica del choque entre la bola y la pared se domina mejor el juego de raqueta

Jearl Walker

Todo juego de pelota a cuatro paredes, como son la raqueta, el squash y el frontón a mano, requiere del jugador una buena dosis de pericia para calcular ángulos y rebotes. La mecánica del choque determina la dirección bajo la cual la pelota se aleja de la pared. El conocimiento de tal mecánica permite al jugador predecir cómo rebotará una pelota que se aproxima y de que modo deberá golpearla para situarla fuera del alcance de su rival. Para tratar de estos fenómenos voy a recurrir a trucos poco conocidos, relacionados con el tema, cuya demostración puede hacerse con una bola maciza muy elástica que se vende en las tiendas de artículos deportivos.

La elasticidad de esta pelota es punto menos que absoluta. Si se deja caer, bota y retorna casi hasta nuestras manos. (Una que fuera perfectamente elástica subiría hasta su altura inicial.) Posee, además, una superficie rugosa; no resbala, pues, cuando corre por el suelo. A causa de su elasticidad y rugosidad, esta pelota rebota, si sabemos provocarlo, de modos muy sorprendentes.

Al lanzarla oblicuamente hacia abajo, recorre la habitación rebotando en una sucesión de saltos altos y cortos y saltos bajos y largos, alternados y repetidos. Si se le da un poco de efecto, se pone a botar a izquierda y derecha, hasta agotar su energía. Pero la demostración más asombrosa y llamativa se consigue arrojándola debajo de una mesa. Una pelota lisa rebotará entre la mesa y el suelo hasta llegar al lado opuesto de aquélla. Nuestra pelota rugosa rebota retornando hacia el lanzador.

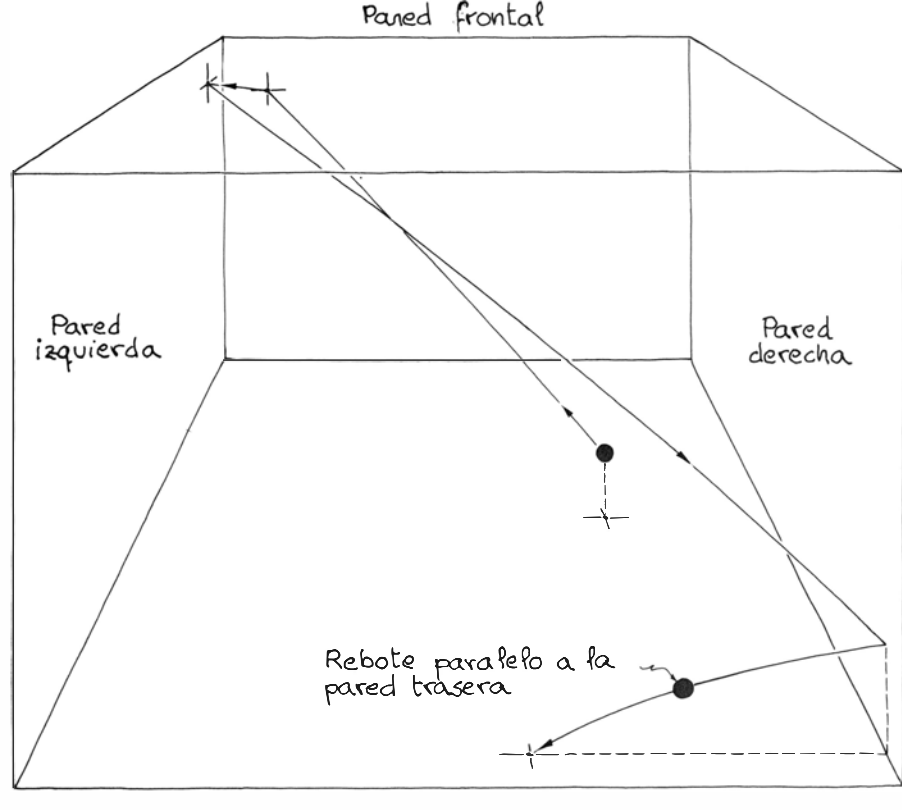
Para estudiar el choque de una pelota con una pared empecé considerando el bote de una pelota maciza y homogénea sobre el suelo. Supongamos que la bola se acerque a éste moviéndose hacia abajo y hacia la derecha. Para facilitar la descripción vamos a descomponer la velocidad en dos partes, una paralela al suelo y perpendicular al mismo la otra. A la primera la llamaremos componente tangencial y, a la segunda, componente normal. La pelota puede girar también en torno a su centro. Esta rotación la consideraremos negativa cuando tenga el sentido del giro de las agujas del reloj, y diremos que la pelota lleva un espín (efecto) horario. Y, al revés, cuando el giro sea opuesto al de las agujas de reloj, la rotación la consideraremos positiva, y diremos que la pelota lleva un espín antihorario.

La energía cinética de la pelota consta de tres partes, una por cada componente de la velocidad y una tercera por el espín. Si la bola fuese perfectamente elástica, en la colisión no variaría la energía cinética total. (Aquí se dice que la energía cinética total se conserva.) Pero sólo una pelota ideal, en un choque ideal, cumple esta condición. En la realidad se pierde algo de la energía cinética por conversión en otras formas de energía; por ejemplo, en vibraciones de la pelota. Pasaremos por alto tales pérdidas y nos ceñiremos a los movimientos de una pelota totalmente elástica.

El choque de la pelota con el suelo cambia la velocidad normal de una manera bastante simple: invirtiendo su sentido pero sin afectar ni a su módulo ni a la energía cinética asociada. La componente tangencial y el espín cambian de una forma más complicada. Ni siquiera así varía la energía cinética total. En un choque elástico puede disminuir el espín, pero entonces la velocidad tangencial aumentará exactamente lo suficiente para que la energía cinética total permanezca constante. Esta condición de conservación de la energía cinética total es una potente herramienta para predecir el retroceso.

Otro extremo importante es que se conserva el momento cinético total. Este recibe una contribución del espín que es igual a la velocidad de giro multiplicada por el momento de inercia de la pelota. Se considera que el momento cinético del espín es negativo si éste es horario, y positivo si es antihorario. El momento de inercia depende de la masa de la esfera y del modo en que la misma está distribuida. Para una pelota maciza y homogénea el momento de inercia vale dos quintos del producto de la masa por el cuadrado del radio.

La otra parte del momento cinético



1. El embarazoso tiro en **Z**

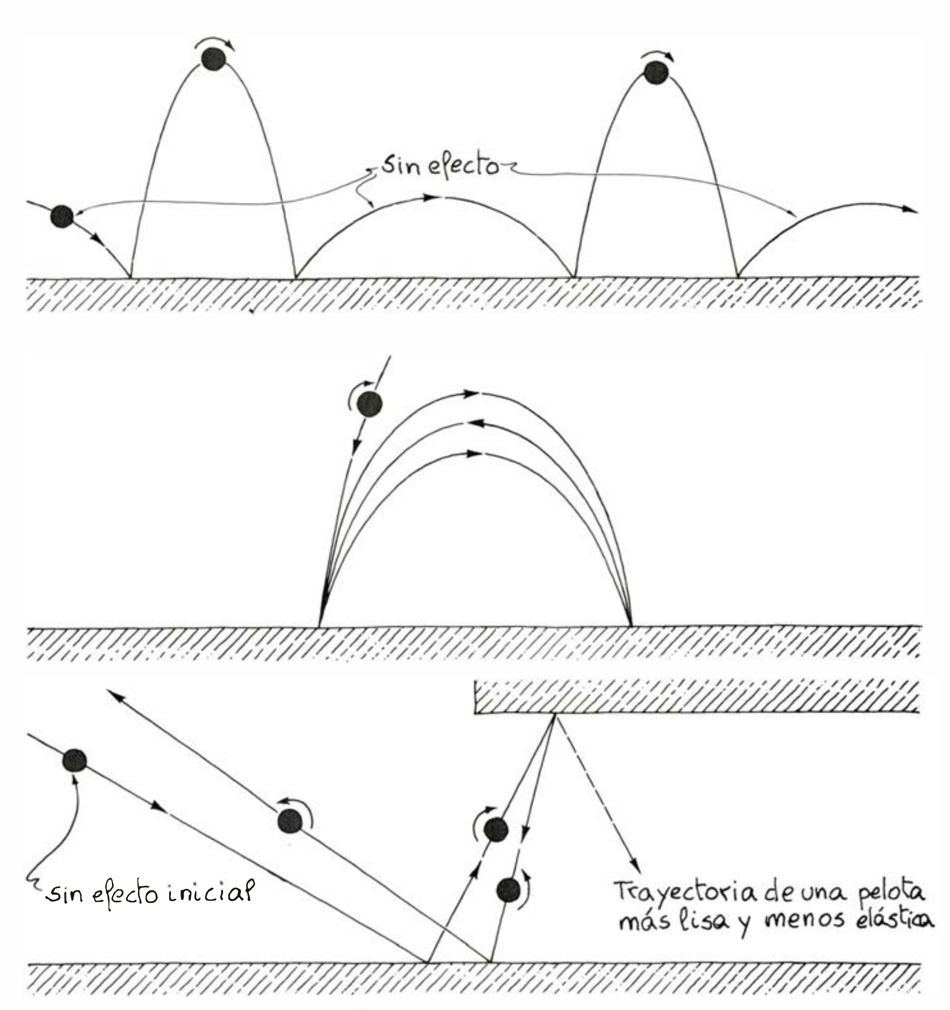
depende de la rapidez con que la pelota se mueve paralelamente al suelo en el instante del contacto. Esta contribución al momento cinético es igual al producto de la masa de la bola por la velocidad tangencial y por el radio. Si la velocidad tangencial está dirigida hacia la derecha, la contribución es negativa; si lo está hacia la izquierda, es positiva. El choque puede alterar ambas contribuciones al momento cinético tanto en módulo como en signo, pero el momento cinético total no varía. En suma, con independencia de cómo se lance la pelota y cómo gire ésta, en un choque perfectamente elástico deben permanecer constantes la energía cinética total y el momento cinético total.

La demostración más sencilla consiste en dejar caer al suelo la pelota. Si ésta carece de espín (efecto) inicial, debe botar y regresar a nuestras manos sin espín, según exigen las leyes de conservación. La única energía cinética que posee la bola es la asociada a su velocidad normal. Como el choque se limita a invertir el sentido de la misma, sin que cambie su módulo, la energía cinética no varía. Y de ésta no puede traspasarse cantidad alguna al espín ni a la velocidad tangencial, con lo que la esfera debe moverse en línea recta hacia arriba. Se cumple asimismo con la condición de conservación del momento cinético, que es nulo antes y después del choque.

Supongamos que le comunicamos un espín horario. Entonces, con el choque, la pelota cambiará de trayectoria. En su contacto con el suelo, el espín crea una fuerza de rozamiento, apuntada a la derecha, la cual invierte el sentido de giro. Además, en virtud de esa fuerza de rozamiento la pelota adquiere una velocidad tangencial, por lo que botará hacia la derecha. La energía para esta velocidad tangencial corre a expensas de la energía del espín inicial.

Cuando la pelota se lanza hacia el suelo inclinadamente y sin efecto se produce también transferencia de energía. Hubiera yo esperado que, tras el bote, la trayectoria tuviese la misma pendiente que la inicial, pero resulta más empinada porque la colisión reduce la velocidad tangencial, convirtiendo parte de su energía cinética en energía de espín. En términos de momento cinético, se reduce el asociado a la velocidad tangencial y aumenta (desde cero) el asociado al espín. Pero se conservan la energía cinética total y el momento cinético total.

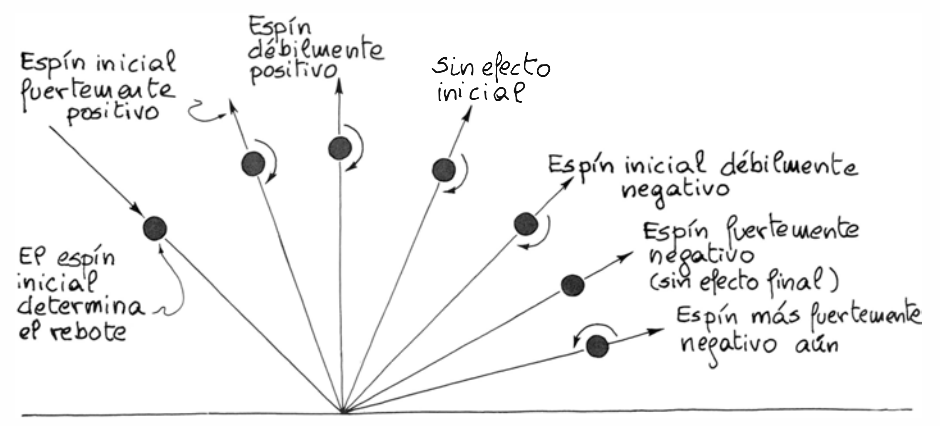
Cuán empinada se muestre la trayectoria tras el choque dependerá de su inclinación inicial y el espín. Cuando el espín inicial es negativo (horario), la pendiente final es menor que cuando la bola se arroja sin efecto. Un espín muy fuerte la envía a lo largo de una trayectoria muy pegada al suelo. Cuando el espín inicial es positivo (antihorario), el bote induce en la pelota una trayectoria empinada, hacia arriba perpendicularmente al suelo, e incluso hacia atrás, dependiendo de la intensidad del espín inicial. El bote la enviará en línea recta hacia arriba si el espín positivo inicial es precisamente el necesario para ello. (El producto del espín por el radio debe ser igual a tres cuartos de la velocidad tangencial inicial.) Si el espín antihorario es mayor, la pelota retrocede hacia la izquierda. Si es menor que aquel valor límite, nulo o negativo (horario), el retroceso es hacia la derecha.



2. Extraños rebotes de una pelota rugosa y elástica

La inclinación del bote puede predecirse teniendo en cuenta el frotamiento en el punto donde la pelota roza al suelo. El sentido de esta fuerza es opuesto al del movimiento de la superficie de la pelota. En el momento del contacto, el movimiento de esta superficie tiene sus orígenes en la velocidad tangencial y el espín. Y el frotamiento se opone a la suma de ambos movimientos. Por ejemplo, si la pelota se lanza hacia abajo, con inclinación y sin efecto, la superficie se moverá hacia la derecha al tocar el suelo. La fuerza de rozamiento que actúa sobre esa superficie se dirigirá hacia la izquierda, y así se reduce la velocidad tangencial. La esfera botará hacia la derecha con una velocidad en esa dirección menor que la de antes del choque. Por no afectar el rozamiento a la velocidad normal, la trayectoria de retroceso será más empinada que la de aproximación.

Me ocupé también de lo que pasa cuando la pelota brinca repetidamente en el suelo. Lancémosla hacia la derecha y sin efecto. Con el primer bote se invertirá la velocidad normal (por lo cual la bola sube), disminuirá la velocidad tangencial y aparecerá un espín horario. Luego, la pelota se elevará hasta la altura máxima y regresará al suelo. Lo notable y sorprendente de este rebote es que restituirá el espín inicial (que sería nulo) y la velocidad inicial. El resultado será el mismo, con inde-



3. Influencia del espín inicial sobre el retroceso

pendencia de los valores iniciales del espín y la velocidad tangencial. Y si la pelota continuara botando por el suelo, el espín y la velocidad tangencial adquirirían sus valores iniciales a cada número par de botes.

Este fenómeno se evidenció claramente en el movimiento de la pelota de juguete. El ecuador de ésta lo pinté con el propósito de observar el espín. Al lanzarla al suelo sin efecto inicial, el primer salto fue alto y corto, por lo que su desplazamiento horizontal, antes del segundo salto, no resultó muy grande. El segundo salto fue bajo y largo, y con la pelota prácticamente sin espín. A partir de entonces se repitió la secuencia de un salto alto y corto seguido de otro bajo y largo. Como la pelota no era totalmente elástica, cada salto tenía menos energía que el precedente. Una esfera perfectamente elástica recuperaría periódicamente su espín nulo inicial y su velocidad tangencial inicial.

Del inesperado comportamiento de una pelota arrojada bajo una mesa, de suerte tal que golpee la cara inferior del tablero, responde la interacción entre el espín y la velocidad tangencial. Si la pelota carece de efecto inicial, retrocederá desde el suelo a lo largo de una trayectoria empinada y dotada de un espín horario rápido; cuando dé en la mesa, rebotará hacia la izquierda con un espín antihorario. El segundo salto desde el suelo será también con un espín antihorario. Aquí, la velocidad normal se habrá invertido tres veces, pero sin cambiar de valor. Por su parte, la velocidad tangencial apuntará entonces hacia la izquierda y su valor habrá variado muy poco. O sea, la pelota retornará al punto de lanzamiento.

Supongamos que la pelota fuese menos rugosa y menos elástica. En el primer bote tendríamos un espín débil y en el segundo (en la cara interna del tablero) la pelota no se dirigiría hacia la izquierda. Así, proseguiría desplazándose hacia la derecha, hasta agotar su energía cinética.

Seguidamente volví mi atención hacia una pelota hueca e idealmente elástica, de las que se usan en el juego de raqueta. Con una bola así deben poderse realizar las mismas tretas que con otra maciza, aunque difieran los valores del espín al ser distinto su momento de inercia. Lancémosla inclinadamente hacia el suelo (hacia la derecha): brincará recta hacia arriba, si el espín es antihorario y su producto por el radio vale un cuarto de la velocidad tangencial, y no tres cuartos, como antes.

En la raqueta, el servicio procede de la pared delantera de la cancha. La pelota retrocede hacia el adversario, bien directamente, o bien rebotando en las paredes laterales. El adversario debe devolverla a la pared frontal, antes de que rebote dos veces en el suelo. Salvo en el saque, la pelota puede también hacerse incidir en la pared trasera y en el techo. Voy a considerar los golpes permitidos después del saque.

Sólo hay dos maneras de que el jugador dé efecto a la pelota con su raqueta: golpeándola hacia adelante y por arriba (logrando el efecto de sobregiro) o golpeándola hacia adelante y por abajo (logrando el efecto de contragiro). En la figura 6 se representan los espines correspondientes vistos desde el costado izquierdo de la cancha.

Consideremos una pelota que haya recibido un golpe fuerte y que, a poca altura, se dirija con sobregiro hacia la pared frontal. Este choque será análogo al que les he descrito para una pelota maciza. El sobregiro (un espín horario en la figura) crea una fuerza de rozamiento apuntada hacia arriba, en cuya dirección dirige la pelota e invierte el espín. Cuando la misma regresa al suelo, el espín antihorario obliga a que el rebote sea bajo y hacia la derecha de la cancha. La ventaja potencial de este tiro estriba en que el contrincante acaso no espere un rebote alto en la pared delantera ni un salto bajo en el suelo.

Si la pelota se golpea fuerte y a poca altura, impulsándola con contragiro, o espín antihorario, hacia la pared frontal, rebotará hacia el suelo con un espín horario. El impacto en el suelo estará cerca de la pared frontal y será empinado hacia arriba. La ventaja potencial de este tiro estriba en que el contrincante acaso no pueda llegar a la pelota antes de que dé, por dos veces, en el suelo.

No acostumbro a comunicar, de salida, efecto a la pelota, o le comunico muy poco, pero termina por adquirirlo en cuanto rebota en una pared o en el techo. Imaginemos un tiro al techo, como los que suelo hacer para variar el ritmo del partido. Aquí mi rival debe adaptarse, no sólo a la nueva trayectoria, sino también a unos inesperados saltos en el suelo. Supongamos que haya hecho que la pelota bote en la pared frontal hacia el techo. De éste saldrá con un espín horario y, cuando rebote en el suelo, su velocidad tangencial se reducirá acusadamente, haciendo que salte derecha hacia arriba. Entonces, mi rival, que estará esperando una trayectoria de retroceso similar a la de acercamiento al suelo, quedará muy retrasado en la cancha. (Los pelotaris, a las jugadas en las que se consigue dejar al contrario demasiado retrasado para alcanzar a devolver la pelota, las llaman, en general, "dejadas".)

Haciendo que la pelota rebote en el techo hacia la pared frontal, al suelo se acercará dotada de un espín antihorario. Al chocar con el suelo aumentará su velocidad tangencial y dará un salto bajo. Y mi rival volverá a equivocarse al estimar la trayectoria de retroceso y perderá esa pelota. Estos dos tiros al techo mejoran si se lanzan, más o menos, desde el centro de la cancha, pues así el espín, cuando la pelota se aproxima al suelo, es mayor y se acentúa la dejada.

Supongamos que la pelota haya pegado en la pared delantera de tal modo que vuelva hacia el lado izquierdo de la cancha. Mirando desde arriba, y prescindiendo de que la gravedad curve la trayectoria hacia abajo, la situación es análoga a la de una pelota maciza lanzada inclinadamente hacia el suelo. En la colisión se invierte la velocidad normal (perpendicular, en este caso, a la pared frontal), disminuye la velocidad tangencial (dirigida hacia la pared izquierda) y aparece un espín horario. A causa de la disminución de la velocidad

tangencial, la trayectoria, vista desde arriba, será más empinada que la inicial con respecto a la pared frontal. En el juego de raqueta, el adversario puede aprender enseguida cómo hacer frente a este tipo de rebotes.

Un tiro al que cuesta más anticiparse es el que rebota en dos paredes. Imaginemos, visto desde arriba, un lanzamiento en el cual la pelota bote en la pared frontal y, seguidamente, rebote en la izquierda. En el primer impacto la bola adquire un espín horario y una velocidad dirigida hacia la pared trasera. ¿Podemos hacer que la pelota rebote en la pared lateral en cualquier dirección que elijamos, o bien está ya determinado el ángulo de retroceso? ¿Puede ser nulo el espín final, o de cualquier valor, horario o antihorario? Para responder a estas preguntas me valí de los cálculos publicados independientemente por Richard L. Garwin, de la Universidad de Columbia, y George L. Strobel, de la de Georgia.

Supongamos una pelota lanzada hacia la pared frontal sin efecto y con una velocidad normal inicial pequeña. Este disparo puede hacerse desde un punto contiguo a la parte delantera de la pared lateral derecha. Entonces, una pelota idealmente elástica rebotará en la pared lateral izquierda bajo un ángulo de unos doce grados. Lanzando desde una posición más cercana al centro de la cancha, la velocidad normal inicial será mayor y el ángulo de rebote en la pared lateral izquierda será menor: la pelota viajará hacia la parte posterior de la cancha, a lo largo de la pared.

¿Cómo sacarle partido a esta combinación? Supongamos que nuestro contrincante se halle hacia el centro de la pared derecha. Haciendo que la pelota bote en la pared delantera y rebote en la izquierda, de modo que se dirija hacia atrás ceñida a ésta, será casi imposible que pueda devolvernos el golpe. Y aun cuando no se encuentre lejos del trayecto final de la pelota, el rebote en la pared lateral le resultará, por lo menos, desconcertante.

Al comparar mis cálculos con movimientos reales de pelota de raqueta comprobé que concordaban bastante. El ángulo de retroceso más inclinado en la pared lateral se mostró superior a los doce grados. A medida que incrementaba la velocidad normal inicial, desplazándome desde la pared derecha hacia el centro de la cancha, dicho ángulo disminuía hasta que la pelota se pegaba, casi, a la pared en su camino de vuelta.

La discrepancia entre la realidad y lo predicho en el rebote en la pared lateral hay que atribuirla a la naturaleza inelástica del choque de una pelota real. Si ésta incide frontalmente sobre una pared, se comprime uniformemente y almacena su energía en forma de energía potencial elástica. De ésta sólo una parte se reconvierte en energía cinética cuando la pelota se aparta de la pared, y recupera su forma esférica. Una pelota de raqueta puede retroceder con el 60 por ciento de su energía en una colisión así. En tal circunstancia su velocidad normal sería del orden del 80 por ciento de su valor inicial. (El cambio de velocidad es proporcional a la raíz cuadrada de la variación de energía.)

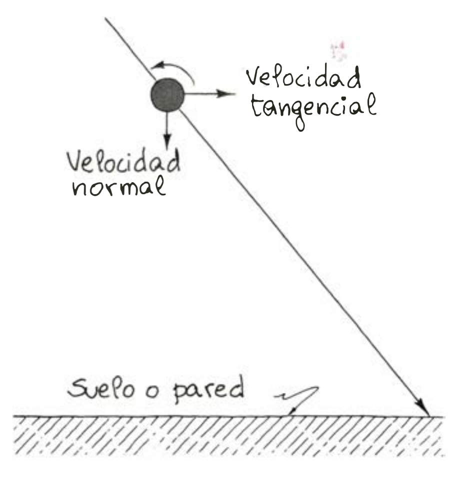
Más difícil es interpretar un choque oblicuo, pues la compresión no es uniforme y depende del ángulo de incidencia. La pérdida de energía cinética y momento cinético reduce tanto el espín como la velocidad tangencial. (Cuando la pelota vuela a ras del suelo o la pared, en un tiro muy oblicuo, esa pérdida de energía puede percibirse como un sonido muy agudo en el momento en que la pelota golpea la superficie.) Para mis cálculos me pareció adecuado reducir en 0,4 el espín y la velocidad tangencial tras el choque. Así conseguí un acuerdo mayor entre las predicciones y la realidad.

¿Por qué no regresa hacia mí la pelota de raqueta real que lanzo bajo una mesa? Porque la reducción en la energía y momento cinético en los rebotes en el suelo y en la cara interior de la mesa constriñen a la bola a rebotar casi verticalmente hasta agotar su energía cinética.

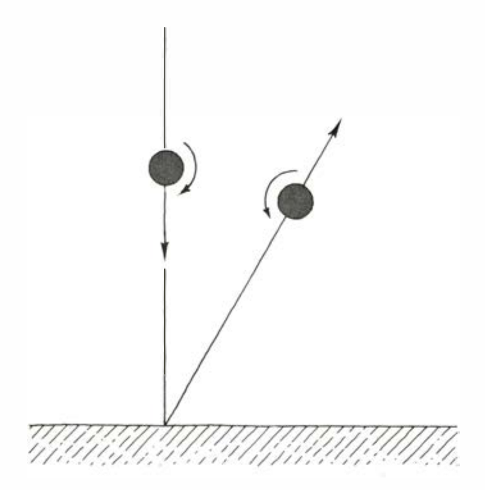
¿Hay manera de asestar un golpe a una pelota contra la pared frontal de modo que rebote en una lateral paralelamente a la frontal? Con un tiro así podrían ganarse casi todos los partidos, pues el rival no llegaría seguramente a tiempo a la pelota. Pero resulta que tal giro es imposible; los rebotes laterales se dirigen siempre hacia atrás.

¿Puede tener un espín de cualquier dirección, e incluso un espín nulo, una pelota rebotada? Sí, puesto que su espín final depende del cociente inicial entre sus velocidades normal y tangencial. Para una pelota de raqueta perfectamente elástica resulta un espín nulo cuando ese cociente vale entre 1 y 5. Si vale menos, el espín es horario (mirando desde arriba); si vale más, el espín es antihorario.

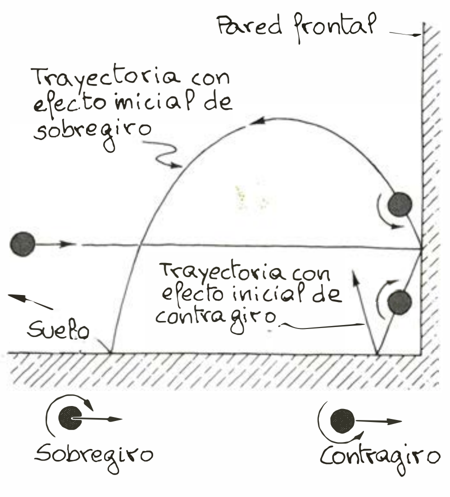
El tiro en *Z* es un rebote a tres paredes que, al verlo, parece un prodigio. Cuando empezó a introducirse a comienzos de los setenta, desorientó hasta a los jugadores más experimentados. Aquí la pelota se lanza contra la parte superior izquierda de la pared frontal, desde donde bota hacia la pared izquierda, cruza entonces la can-



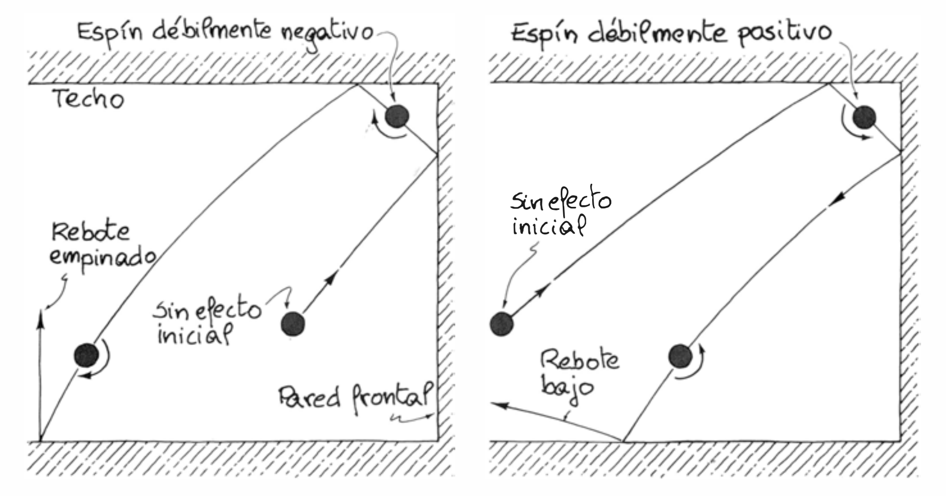
4. Las dos componentes de la velocidad de una pelota



5. Así desvía un rebote el espín

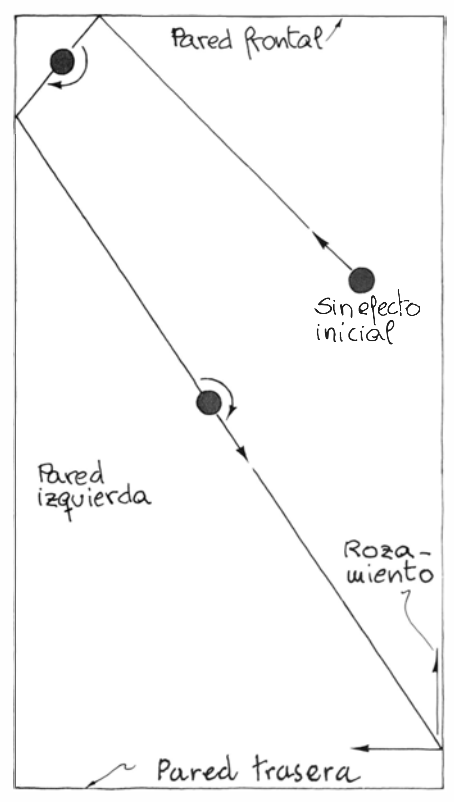


6. Energética del sobregiro y el contragiro



7. Maneras de aprovechar el techo

cha en dirección a la parte posterior de la pared derecha y en ésta rebota paralelamente a la pared trasera. Para anticiparse a este rebote final, todo contrincante necesitará gran experiencia; pero aún así le será difícil devolver la pelota a la pared delantera. Si el golpe en *Z* no se lanza bien, la pelota puede aún ser de difícil devolución si golpea en el suelo y luego en la pared posterior. Efectivamente, en tal caso el adversario deberá alcanzarla junto a la pared posterior antes de que rebote por segunda vez en el suelo.



8. Tiro en **Z** *visto desde arriba*

Al principio imaginaba la imposibilidad de un tiro en *Z* perfecto. Dudaba que el último rebote pudiera hacerse paralelo a la pared trasera. Entonces, provisto de mis matemáticas, me puse a seguir los saltos.

No tardaron en presentarse las dificultades. Si se admite que la pelota es perfectamente elástica, en la pared izquierda rebotará bajo un ángulo tan pequeño que irá a dar en la pared trasera y no en la derecha. Por ello, como dato para mis cálculos tomé una cancha muy corta. Desprecié, además, la curvatura debida a la gravedad e hice los cálculos como si la pelota se moviera en un plano paralelo al suelo.

Para lanzar un tiro en *Z* el jugador se coloca junto a la pared derecha, aproximadamente hacia la mitad de la cancha. La pelota hay que enviarla a la parte superior izquierda de la pared delantera, a un metro, más o menos, de la esquina y a la misma distancia aproximada del techo. Como la pelota abandona, así, la pared izquierda con un espín horario, su choque con la pared derecha crea una fuerza de rozamiento apuntada hacia la pared frontal.

Consideremos la velocidad y el espín de la pelota inmediatamente antes e inmediatamente después del choque con la pared derecha. La velocidad normal se invierte y envía la pelota hacia la otra pared lateral. ¿Qué le ocurrirá al espín y a la velocidad tangencial? Este choque es similar a otro que hemos examinado antes. El sentido del rozamiento durante el mismo es opuesto tanto al espín como a la velocidad tangencial, con lo que ésta disminuye y el primero se invierte. En las condiciones adecuadas, la velocidad tangencial podrá anularse y la trayectoria de la pelota será perpendicular a la pared lateral. Así es como un tiro en *Z* perfectamente ejecutado hace que la pelota se mueva paralelamente a la pared trasera.

Al incluir en mis cálculos la pérdida de energía en cada colisión obtuve unas predicciones que se aproximaron más a la trayectoria real de un tiro en *Z* en una cancha de las dimensiones correctas. Pero siguió presente la posibilidad de que el rebote final fuese paralelo a la pared posterior. Eran, sin embargo, cálculos incompletos; la trayectoria real es tridimensional. Con la hipótesis de trayectoria plana los había simplificado, ya que el eje a cuyo alrededor gira la bola se mantiene paralelo a la pared; lo cierto es que suele formar un ángulo no nulo con la pared lateral.

En el tiro a tres paredes se pega, obviamente, a las tres. La pelota bota desde la pared lateral derecha hacia la delantera, donde rebota hacia la izquierda. Este golpe está pensado para confundir al contrario, pero si la bola va a parar al centro de la cancha, aquél puede tener una probabilidad no pequeña de devolverla a la pared delantera. Me planteé la cuestión de si era posible organizar ese tiro de modo que la pelota rebotara en la pared izquierda paralelamente a la frontal. Así, al esperar que la pelota fuera hacia la parte de atrás de la cancha, el contrincante se vería seguramente sorprendido por el inesperado rebote.

Este tiro lo ensayé sin suerte de múltiples formas. Preguntándome si el problema no estaría en mi falta de destreza recurrí de nuevo a las matemáticas. De acuerdo con mis cálculos, ese rebote es posible si la pelota sale con gran energía y forma un ángulo reducido con la pared derecha. De haber empezado por los cálculos me habría ahorrado no pocos raquetazos en vano.

Son muchos los tiros, con pelota maciza o hueca, susceptibles de estudiarse. Y acaso queden algunos tiros ingeniosos que los jugadores profesionales hayan aún de descubrir. Puede ser interesante estudiar de qué modo pierde energía una pelota al chocar oblicuamente con una pared. También puede serlo seguir el vuelo de una pelota en tres dimensiones, sin que el eje del espín sea ya paralelo a las paredes. A este fin puede resultar muy útil idear un modelo de la pelota para el ordenador. Si alguien experimenta con una pelota maciza y de gran elasticidad, que tenga cuidado. Esto lo he probado sólo una vez y la pelota se movía y rebotaba con tal rapidez que me obligaba a apartarme de su camino.

Libros

De la razón, o inteligencia natural, a los algoritmos, o inteligencia artificial

Diego Gracia y José Sanmartín

INTELIGENCIA SENTIENTE, por Xavier Zubiri. Madrid, Alianza Editorial Sociedad de Estudios y Publicaciones. Vol. I: *Inteligencia y realidad*, 1.ª ed. 1980, 2.ª ed. 1982 (ambas con el título de *Inteligencia sentiente*), 3.ª ed. 1984 (con el título actual). Vol. 2: *Inteligencia y logos*, 1982, Vol. 3: *Inteligencia y razón*, 1983. Meses antes de su fallecimiento, ocurrido en Madrid el 21 de septiembre de 1983, terminaba Zubiri la publicación de su trilogía sobre el proceso de la intelección humana. Es, ciertamente, su obra maestra, y forma junto con *Sobre la esencia*, aparecida en 1962 y reeditada varias veces, el núcleo original del pensamiento filosófico de su autor. Durante muchos años Zubiri ha gozado de un enorme prestigio social, como gran promesa de la filosofía. Pero esa época ha pasado definitivamente, Zubiri ya no puede prometer nada, y debe ser juzgado por su obra filosófica. Por fortuna alcanzó a publicar en vida estos tres volúmenes, que encierran el mensaje nuclear de su filosofía y la clave interpretativa de todos sus demás escritos.

En el intento por situar esta obra de Zubiri en el contexto de la producción filosófica usual, habría que decir que es un tratado de teoría del conocimiento. Sin embargo, si alguna intención rectilínea ha tenido su autor a lo largo de las más de mil páginas de la trilogía, ha sido demostrar la inconsistencia de ese nutrido género ligerario. Lo anuncia ya en el prólogo del primer volumen: "La presunta anterioridad crítica del saber sobre la realidad, esto es sobre lo sabido, no es en el fondo sino una especie de timorato titubeo en el arranque mismo del filosofar. Algo así como si alguien que quiere abrir una puerta se pasara horas estudiando el movimiento de los músculos de su mano; probablemente no llegará nunca a abrir la puerta. En el fondo, esta idea crítica de anterioridad nunca ha llevado por sí sola a un saber de lo real, y cuando lo ha logrado se ha debido en general a no haber sido fiel a la crítica misma".

Desde el arranque mismo de su investigación Zubiri declara una y otra vez que no intenta otra cosa que "describir" el acto mismo de aprehender las cosas que tiene el hombre. El mal secular de las doctrinas del saber y de las teorías del conocimiento ha estado en no describir, en interpretar, teorizar. Toda la filosofía clásica, de Parménides a Descartes, estuvo basada en una interpretación muy particular, y desde luego discutible, de qué sea conocer. Se partió de la idea de que los sentidos nos dan lo externo y mudable de la cosa, y que sólo la inteligencia es capaz de penetrar los accidentes y llegar al conocimiento de la esencia. Se trata de una teoría dualista de carácter metafísico, que disocia el sentir y el inteligir humanos, en contra precisamente de lo que dicta la experiencia. El resultado es lo que Zubiri llama "inteligencia concipiente", origen del conceptismo metafísico, una de las formas históricas del idealismo filosófico.

La modernidad, sobre todo la nueva física de Galileo y Newton, conmovió de raíz todo ese edificio y obligó a los filósofos a emprender nuevas vías. A la metafísica de la inteligencia va a suceder la teoría del conocimiento, que alcanza su madurez en la obra de Kant. El primer problema que tiene que resolver el filósofo, piensa Kant, es saber si la metafísica reúne las condiciones exigidas por el conocimiento científico. Para ello, Kant hace un análisis de las condiciones de posibilidad del conocimiento científico en las dos primeras partes de la *Crítica de la razón pura*. En la primera, la Estética transcendental, define las categorías de la sensibilidad humana, siguiendo el modelo de la ciencia matemática. Después, en la segunda parte, la Analítica transcendental, tomando como guía la nueva física, establece el sistema de categorías del entendimiento. La sensibilidad es intuitiva, en tanto que el entendimiento elabora conceptos. El conocimiento objetivo o científico se consigue por la síntesis de los datos de la sensibilidad y las categorías del entendimiento. Tras esto, en la tercera parte, la Dialéctica transcendental, Kant se pregunta por la objetividad o cientificidad del conocimiento metafísico, y naturalmente responde de modo negativo. Frente a la metafísica se levanta, pues, la teoría del conocimiento. Pero si bien ésta tuvo el mérito de desenmascarar la inconsistencia del saber metafísico, Zubiri piensa que cayó en una trampa no menos perniciosa. En vez de analizar lo más fielmente posible el proceso de la intelección humana, Kant construyó de nuevo una ingente teoría, que además seguía siendo dualista, al establecer una cesura insalvable entre el sentir y el entender humanos. El resultado es lo que Zubiri llama repetidamente en el libro "inteligencia sensible", raíz de todo el idealismo transcendental de la época moderna.

¿Es posible ir más allá? ¿Podrá superar la filosofía tanto la etapa metafísica como la etapa crítica y situarse en un plano más radical? Por lo menos hay que intentarlo, piensa Zubiri, y a ello dedica su esfuerzo. Ciertamente, no ha sido el único ni el primero en proponerse esta meta, ni ha caminado solo. Si algo se propuso Husserl con su *Phänomenologie* fue situar el problema del conocimiento a un nivel previo y más radical que el de la *Erkenntnistheorie*, mediante la descripción de los hechos en tanto que datos de conciencia. Zubiri fue un joven discípulo de Husserl, y asistió asombrado al titánico esfuerzo de éste por situar adecuadamente el problema de la filosofía. Un esfuerzo que acabó en otro nuevo idealismo, precisamente porque la descripción se hacía sobre los datos de la llamada conciencia pura. Ortega vio pronto este peligro, ya evidente en el primer volumen de *Ideen*, publicado en 1913, y al año siguiente intentó rectificarlo en *Meditaciones del Quijote*, desviando el punto de mira de la descripción fenomenológica de la conciencia pura a la vida humana, que adquiere así el rango de realidad radical, y pasando a ocupar el yo y la circunstancia los lugares que Husserl asignaba a la noesis y al noema. Un tercer maestro de Zubiri, Heidegger, intentó soslayar el idealismo husserliano haciendo de la noesis el hombre entero, el *Dasein*, y del noema el mundo, o *In-der-Welt-sein*. El punto de confluencia es, naturalmente, el *Sein* o Ser, que pasa así a ocupar el puesto que en Husserl tenía la conciencia y en Ortega la vida humana. Bien entendido, que con esto Heidegger no retornó a la metafísica clásica, que para él casi nunca abandonó el plano meramente óntico para elevarse a la ontología.

El pensamiento de Zubiri no se entiende más que situado en esta línea. Su novedad está en haber centrado el análisis no en la conciencia, ni menos

en la vida o en el ser, sino allí donde de hecho se produce la aprehensión humana, en el sentir. Y como el sentir humano es siempre material, resulta que el punto de arranque de la descripción zubiriana es cualquier cosa menos idealista, es materialista o, como él prefería decir dado el lastre histórico del término, materista. Por los sentidos el hombre aprehende cosas materiales. Cosa tiene aquí un sentido sumamente amplio, equivalente a nota, no lo que la metafísica clásica entendía por sustancia, ni lo que la filosofía moderna llamaba objeto. Pensemos, por ejemplo, en el calor. El hombre aprehende el calor por sus sentidos. Y lo aprehende de una forma muy peculiar, si se compara, por ejemplo, con lo que parece sucederle al animal. En el animal, según parece, el calor se agota en ser estímulo calentante que provoca una respuesta objetiva (p. e. de huida). En el animal las cosas se agotan en ser estímulos objetivos que desencadenan una respuesta. Pero en el hombre no sucede así. Para el hombre el calor no sólo "está calentando" sino que "es caliente". Lo cual no significa necesariamente que el calor sea "propiedad" de una "cosa" en sí caliente, ya que esto supondría la caída en una ingenua metafísica realista, en contra de todo lo anunciado. Significa tan sólo que el calor se presenta como siendo caliente "en propio" o "de suyo". Y ello aunque se agotara en el acto de aprehensión y no tuviera ninguna otra realidad ni antes ni después ni fuera de él. En la aprehensión el calor se me actualiza como algo que es "de suyo" caliente. Nada más. No se está hablando de lo que las cosas son "allende" la aprehensión, sino de cómo se actualizan "en" la aprehensión. Y lo que se dice es que en la aprehensión se actualizan al hombre de un modo distinto a, por ejemplo, al animal. Al animal se le actualizan, a lo que parece, como estímulos objetivos y nada más; al hombre, por el contrario, se le actualizan como notas "en propio" o "de suyo". Pues bien, esto es lo que Zubiri entiende por "realidad". La realidad es la formalidad como las cosas se le actualizan al hombre en la aprehensión, por tanto, el "de suyo", a diferencia de otras formalidades, como por ejemplo la del animal, que parece ser sólo de "estimulidad". Para el animal las cosas son estímulos objetivos. Para el hombre las cosas son realidades.

De este principio Zubiri saca un buen número de consecuencias. Primera, que realidad no es sinónimo de cosa "en sí", como defendió siempre la metafísica, ni tampoco de cosa "en mí", como pensó el subjetivismo moderno. Realidad es algo previo al en sí y al en mí, es "de suyo", si se prefiere, "de sí". Segunda consecuencia: esta formalidad del "de suyo" o de "realidad" define el sentir humano, a diferencia del sentir animal, como "sentir intelectivo" o "inteligencia sentiente". Frente a la "inteligencia concipiente" de la metafísica y la "inteligencia sensible" de la época moderna, Zubiri propone la "inteligencia sentiente". Tercero: esto obliga a definir la inteligencia de modo muy distinto a como hoy es usual. Por inteligencia suele entenderse hoy la capacidad de procesar información, esto es, estímulos objetivos. En este sentido es obvio que además de la humana hay inteligencia animal y aún inteligencia artificial. Zubiri, sin embargo, reserva el término inteligencia para el modo de procesar información específicamente humano, actualizando las cosas como "de suyos" o "realidades". Cuarto: en la aprehensión el hombre actualiza la cosa como siendo tal cosa, p. e. color, pero a la vez como siendo real; por tanto, hay un doble momento en la aprehensión, el "talitativo" (el color o la piedra como tales color o piedra) y el "transcendental" (el color y la piedra como simplemente reales). Quinto: cuando lo aprehendido no es una realidad simple, una nota, como el color, sino una realidad compleja que tiene muchas notas, por ejemplo la piedra, ésta se me actualiza como un conjunto de notas que forman un "de suyo", por tanto como una estructura clausurada de notas, lo que Zubiri denomina una "sustantividad". La sustantividad surge del análisis del modo como las cosas me quedan actualizadas en la aprehensión, a diferencia de la sustancia metafísica, que hacía referencia a la estructura de la cosa en sí.

Los puntos anteriores resumen de algún modo el contenido del primero de los volúmenes de la trilogía, titulado *Inteligencia y realidad*. Una vez precisado este punto de partida, en la exposición de los otros dos se puede ser más breve. Todo el primer volumen está dedicado a estudiar lo que Zubiri llama la "aprehensión primordial", que no constituye más que un momento, el "individual", de la aprehensión humana. Junto a él hay siempre otro que Zubiri llama "campal"; si el primero nos actualiza la realidad de la cosa, nos permite saber que *es real*, el segundo, en el que inteligimos unas cosas "entre" otras, nos actualiza lo que la cosa es *en realidad*. El estudio de la intelección campal es el objeto del segundo volumen, *Inteligencia y logos*. Ese modo de intelección que se denomina logos consiste en la actualización de unas cosas reales respecto de otras, de las otras aprehendidas en el campo. Respecto de las otras cosas reales del campo, cada una es lo que es, por tanto define su realidad. Esta definición es la afirmación, el juicio. Para juzgar lo que es una cosa entre otras lo primero que hay que hacer es tomar distancia, retraerse y "pararse a considerar" la cosa desde el campo; el resultado es lo que Zubiri llama "simple aprehensión", a diferencia de la "aprehensión primordial" ya estudiada. Naturalmente, simple aprehensión tiene aquí un sentido completamente distinto al usual en filosofía. Zubiri la define como la libre creación o postulación desde el campo de lo que en la cosa "sería" el contenido de su realidad, en la dirección de su "esto" (percepto), de su "cómo" (ficto) y de su "qué" (concepto). La novela, por ejemplo, se construye básicamente según perceptos y fictos, en tanto que la matemática según conceptos. Pero desde esa distancia y con el utillaje de perceptos, fictos y conceptos, la inteligencia vuelve a las cosas reales de las que ha tomado distancia, ya no en busca de lo que "sería" la cosa real, sino en busca de lo que ella "es" en realidad. Ya no se trata de simple aprehensión sino de "intelección afirmativa". Es el retorno desde el campo a la cosa, lo que lleva ineludiblemente a "juzgar". En el juicio se afirma lo que "es" la cosa "en realidad". Este "es" reúne en la afirmación los momentos del "esto, cómo, qué". En un juicio del tipo este agua está caliente, se presupone que aquello de que se juzga, este agua, es real (aprehensión primordial), y lo que se afirma es que es campalmente caliente (el calor como nota individual estaba ya actualizado en aprehensión primordial, pero no como nota campal, es decir, como "en realidad" caliente a diferencia de lo "en realidad" frío). Juzgar es afirmar lo que la cosa ya aprehendida como real es en realidad. Esta afirmación a veces consigue dar certeza, otras plausibilidad, otras duda, otras sólo barrunto y otras, en fin, ignorancia. Hay afirmaciones de ignorancia tanto como de certeza.

Llegados a este punto conviene recordar que aún seguimos analizando lo que son las cosas "en" la aprehensión, no "allende" la aprehensión. Las afirmaciones y los juicios no son de existencia, sino de realidad, de realidad en la aprehensión. De esta aprehensión humana dijimos que tenía un doble momento, el individual (cuyo correlato es la aprehensión primordial) y el campal (que culmina en el juicio afirmativo y llamamos logos). En la aprehensión primordial se nos actualiza la cosa

como real; en el logos la reactualizamos afirmando lo que es en realidad. Pero en ambos casos se trata de la cosa sólo en tanto que aprehendida, en la aprehensión.

Queda, pues, abierto un último e ingente problema, el del paso desde la realidad dada en impresión a la realidad allende la aprehensión: es la obra de la razón. Ella va a intentar inteligir en las cosas, no que son reales ni lo que son en realidad, sino aquello que son *en la realidad*, por tanto, fuera de la aprehensión, más allá de ella. Se trata, dice Zubiri, de una estricta "marcha" intelectiva desde la aprehensión allende la aprehensión, en busca de lo que las cosas son en la realidad del mundo. En la aprehensión primordial yo veo el color como real y en el logos lo afirmo como verde. En la aprehensión esto es real y verdaderamente así. Pero llega un momento en que tengo que preguntarme por la realidad profunda del color, y entonces la razón científica lo transformará en longitudes de onda, etc. Del mismo modo, el bastón parcialmente sumergido en el agua se me actualiza en la aprehensión en tal forma que el logos lo afirma como roto. Esto es verdad, y si no lo fuera nunca podría irse más allá en la discusión del asunto. Pero la razón científica puede profundizar en la cuestión hasta descubrir las leyes de la refracción, y decir que en la realidad el bastón no está roto; sólo sumergido en dos medios distintos, el agua y el aire. Tal es la tarea ingente de la razón, el análisis en profundidad de lo que las cosas son en la realidad del mundo. Se trata de una tarea fatigosa y siempre inacabada, motivo por el cual Zubiri evita escrupulosamente las típicas expresiones metafísicas de "realidad en sí", "noumeno", etc. Pocas veces podemos estar seguros de que conocemos una cosa real en toda su profundidad. Ciertamente, el "realismo" zubiriano es muy poco ingenuo y bastante escéptico. Sólo afirma una cosa con toda energía, y es que siempre tenemos el bastión inexpugnable del "de suyo". El "de suyo" es el hilo rojo que permanece inalterable en la aprehensión y allende la aprehensión, y que nos permite realizar la marcha de uno a otro. La marcha no es desde la realidad allende ella, sino que es marcha dentro de la realidad (es decir, dentro del "suyo"), buscando el fondo de la cosa. La razón es marcha en profundidad, hacia el fundamento; es intelección fundamental.

Naturalmente, éste es el argumento del tercer volumen, *Inteligencia y razón*. La razón es búsqueda, *intellectus quaerens*. Su actividad es lo que llamamos "pensar". Las cosas nos "dan que pensar", nos ponen en la necesidad de pensar. Su término es "conocimiento". No toda intelección es conocimiento, en contra de lo que afirma la *Erkenntnistheorie*. Ni la aprehensión primordial ni el juicio dan conocimiento; sólo conoce la razón. Realmente, en este volumen es en el que se dan las claves para la crítica de la teoría del conocimiento y de la llamada filosofía de la ciencia, ya que ambas se han planteado los problemas exclusivamente a este nivel. Estos problemas son tres, que Zubiri denomina "objetualidad", "método" y "verdad racional". Sólo mediante el conocimiento racional adquiere la cosa la condición de "objeto". Aquí Zubiri mantiene una lógica discusión con Kant, replanteando completamente el sistema de las categorías. El "método" es vía de conocer, pero no vía del conocimiento sino vía de la realidad; por tanto, no se identifica con "razonamiento" ni con "lógica". Su término es la "experiencia", que Zubiri define como "probación de realidad". Tras el esbozo de posibilidades característico de la razón, hay que volver a la realidad, la cual aprueba o reprueba los esbozos de posibilidades que hemos construido. En el primer caso se produce el encuentro en que consiste la "verdad racional"; en el segundo, el error. Ese encuentro tiene un nombre, "verificación". En la verificación las cosas, lo real, nos está dando (o quitando) la razón. Razón es siempre esbozo de posibilidades, es un esbozo intelectivo de lo que la cosa real "podría" ser como momento del mundo. De ahí la necesidad de que lo real verifique a la razón en la experiencia. La verificación es cualidad esencialmente dinámica: es siempre y sólo, dice Zubiri, "ir verificando". Y este ir verificando es lo que constituye la experiencia. De ahí que la verdad racional sea siempre y a la vez lógica e histórica. Lógica e históricamente se configuran y desarrollan los diversos tipos de razón, entre los que Zubiri incluye, p. e. la razón poética. Pero los tipos fundamentales de razón son dos: la razón científica, que estudia las cosas en su talidad, y la razón metafísica, que principalmente analiza el orden transcendental. Ellas son los principales esbozos teóricos que construye el hombre sobre lo que las cosas reales pueden ser como momentos de realidad.

Visto desde aquí, desde el final, el inteligir queda modalizado como "entendimiento", cuyo acto propio es el "comprender" y que genera un estado, el "saber". Al entender las cosas quedo sabiendo. "Sabiendo ¿qué? Algo, muy poco, de lo que es real. Pero, sin embargo, retenido constitutivamente en la realidad. ¿Cómo? Es el gran problema humano: saber estar en la realidad." Así termina el libro.

La lectura de esta obra, además de larga y difícil, resulta sumamente incómoda. Son tantos los matices del análisis que con frecuencia los árboles no dejan ver el bosque. No es probable que los lectores, por mucho ánimo que tengan al comienzo de su lectura, lleguen al final. Y ello no sólo por defectos en su composición (si se puede hablar de defectos en una obra que no dudo en calificar de monumental y genial), sino también por los malos hábitos que el lector de textos filosóficos, especialmente el lector español, ha ido adquiriendo. La moda del ensayismo y de la filosofía narrativa, casi novelesca, ha facilitado muy probablemente tanto la "composición" de libros filosóficos como su amplia "lectura", pero a la vez ha convertido en excepcionales la "investigación" de altos vuelos y el "estudio" minucioso. Con lo cual la filosofía corre el peligro de perder rigor y degenera fácilmente en sofística. Sabedor, quizá, de que navegaba contra corriente, en las últimas líneas del prólogo se le escapó a Zubiri la única queja del libro. Dice así: "Hoy estamos innegablemente envueltos en todo el mundo por una gran oleada de sofística. Como en tiempos de Platón y de Aristóteles, también hoy nos arrastran inundatoriamente el discurso y la propaganda. Pero la verdad es que estamos instalados modestamente, pero irrefragablemente, en la realidad. Por esto es necesario hoy más que nunca llevar a cabo el esfuerzo de sumergirnos en lo real en que ya estamos, para arrancar con rigor a su realidad aunque no sean sino algunas pobres esquirlas de su intrínseca inteligibilidad". (D. G.)

INTRODUCCIÓN A LA TEORÍA DE LA COMPUTABILIDAD. ALGORITMOS Y MÁQUINAS, por Hans Hermes. Editorial Tecnos; Madrid, 1984. La "teoría de la computabilidad" o "teoría de algoritmos" constituye el núcleo teórico fundamental del conjunto de disciplinas que se superponen en lo que se ha dado en llamar "ciencia del computador".

Por "algoritmo" se entiende, a su vez, una prescripción del orden secuencial en que se ha de realizar una serie de operaciones para la resolución, en número finito de pasos, de un tipo específico de problemas. La ejecución de esas operaciones está totalmente determinada y debe ser realizada sin referencias al bagaje de conocimientos, imaginación o intuición propios del eje-

cutante. "Algoritmo" y "método efectivo" se consideran sinónimos.

Los casos paradigmáticos de métodos efectivos han sido tradicionalmente algoritmos matemáticos. En ese sentido hay que destacar las reglas de al-Khowarizmi (de quien proviene precisamente el nombre de "algoritmo") para ejecutar las cuatro operaciones aritméticas básicas. Otro caso modélico lo constituye el algoritmo de Euclides para hallar el máximo común divisor de dos enteros positivos que no sean relativamente primos entre sí.

Hace mucho tiempo, pues, que se estudian en matemáticas algoritmos. El término algoritmo se usó, sin embargo, en un principio sólo en relación con la resolución de problemas concretos. Esa fue la causa, en buena medida, de que la necesidad de una definición general de esa noción sólo se evidenciara tardíamente –tras el Renacimiento en particular– desembocando en el ideal –tan caro a Leibniz– de construir un algoritmo para resolver cualquier problema matemático. Una nueva versión de esta cuestión sería, a principios de nuestro siglo, el célebre "problema de la deducción" de la lógica matemática: determinar para dos fórmulas cualesquiera φ y ψ de un cálculo lógico si hay una cadena deductiva de φ a ψ.

Se hicieron múltiples intentos para resolver el problema de la deducción. Todos fracasaron. Ello hizo abrigar la sospecha de que el algoritmo deseado no existiera. Pero una cosa es la sospecha y otra bien distinta la afirmación de que ese método efectivo no existe. Esta última presupondrá la realización de una prueba matemática sólo posible, en este caso, si se dispone previamente de una definición formalmente precisa del concepto de algoritmo, que sustituya de modo satisfactorio su noción intuitiva e informal.

En los años 30 se introdujeron algunos conceptos definidos de modo tal que quedaban satisfechos los requisitos estándard de rigor matemático: el de recursividad (de Gödel-Kleene), el de Turing-computabilidad y el de sistema canónico (de Post). Más tarde, en 1951, sería introducido el de algoritmo normal (por parte de Markov). Todos estos conceptos resultaron ser equivalentes.

En 1936, Church aventuró la conjetura de que el concepto de "efectivo" era identificable con el de "recursivo". Dada la equivalencia acabada de apuntar, esta conjetura (la llamada tesis de Church) puede adoptar diversas formulaciones, ninguna de las cuales irá más allá de un desafío hasta el presente siempre perdido: producir algo que sea efectivo y no sea recursivo.

En 1958 y 1961 se publicaron, respectivamente, dos extraordinarios libros de texto sobre teoría de algorimos: *Computability and Unsolvability*, de M. Davis, y *Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit*, del filósofo Hans Hermes.

Hermes (1912), doctorado por la Universidad de Munich en 1937 y habilitado por la de Bonn en 1947, ha sido profesor de las universidades de Münster y Freiburg y profesor invitado en numerosas universidades extranjeras, entre las que destacan Berkeley y Los Angeles. Sin duda alguna se encuentra entre los tres profesores alemanes de mayor influencia personal sobre el desarrollo de la lógica y los fundamentos de las matemáticas en nuestro país (los otros dos serían Paul Lorenzen y Wolfgang Stegmüller). Un buen número de discípulos suyos, profesionales hoy del saber en este área, así lo acreditan con su docencia e investigación; entre ellos, Jesús Mosterín y José Fernández-Prida.

Aufzählbarkeit... apareció en 1965 en versión inglesa bajo el título de *Enumerability, Decidability, Computability* (se hacen pocas alteraciones en el texto original y se corrigen algunos errores). En 1969 se publica la segunda edición

de esta versión inglesa (el cambio principal lo experimenta ahora la exposición de la lógica minimal de Fitch); en 1971, aparece, finalmente, la segunda edición de la versión alemana. Como declara su autor, en este libro se pretende convencer al lector de que los conceptos precisos en él introducidos (en concreto, los de Turing- computabilidad y recursividad) constituyen elucidaciones matemáticas adecuadas del concepto "intuitivo" de algoritmo, eligiendo como punto de partida de la exposición el examen de la máquina de Turing y tratando sobre esa base los conceptos constructivos más importantes.

El cuaderno de filosofía y ensayo *Introducción a la teoría de la computabilidad* constituye una selección de parágrafos de Hermes-1961; en concreto, se trata de los parágrafos 1 a 9 y 30. La traducción es de Manuel Garrido, ayudado por Aránzazu Martín Santos. (Garrido es uno de los pocos lógicos que, entre nosotros, aún sabe hablar de su disciplina usando un castellano elegante y rico.) En esta versión se ha tenido generalmente en cuenta el texto de Hermes-1971, aunque en algunos ejemplos y en el diseño de la máquina universal de Turing se ha dado preferencia a Hermes-1969.

Aparte de los méritos implícitamente constatados en el párrafo anterior, quiero añadir ahora que el traductor –yendo mucho más allá de los límites impuestos por la mera labor de verter un texto a una nueva lengua– ha ayudado al lector con la corrección de erratas del libro de Hermes –algunas importantes, como la cometida en la tabla de la pág. 92 en la que la segunda instrucción del conjunto (ξ) decía en el texto original 9|rO causando los consiguientes despistes.

Lamentablemente se han deslizado, por el contrario, algunas erratas tipográficas en este cuaderno. Sólo quiero citar una que, de seguro, causará algún dolor de cabeza al lector no avisado, dado sobre todo el contexto en que ha acaecido: el tratamiento del teorema de Gödel, de por sí siempre difícil para el lego, por fácil que se pretenda hacerlo. En la pág. 67, línea 14 (empezando por debajo), dice: "lógica de segundo orden, tal que los contenidos de *F* pueden especificar una fórmula *F* dependiente del siste-cible en el sistema de reglas"; debe decir: "lógica de segundo orden, tal que el contenido de *F* puede interpretarse diciendo que *F* no es deducible en el sistema de reglas".

Hans Hermes no es un lógico de gran originalidad, pero ha poseído en altas dosis capacidad de erudición y, sobre todo, facilidad expositiva. Su libro de 1961 es todo un modelo a este último respecto, y, si tuviera que destacar alguna parte de este texto tomando como base esa claridad didáctica, sin duda que elegiría la recogida precisamente en este cuaderno: la referente a máquinas de Turing, cuya introducción progresiva no se puede titular de otra cosa que de paradigmática. En ξ1-ξ9, tras unas reflexiones preliminares –hechas en lenguaje cotidiano– sobre algoritmos en general y máquinas de Turing en particular, se aborda (1.º) la precisión matemática de éstas últimas como matrices, (2.º) las máquinas elementales de Turing y (3.º) las máquinas de Turing especiales; unos cuantos ejemplos sobre las funciones Turing-computables y los predicados Turing-decidibles cierran esta primera tanda de cuestiones. A la máquina universal de Turing, abordada por Hermes en la obra original nada menos que 21 parágrafos más adelante, se la hace "avanzar" en este cuaderno hasta ponerse en contacto con el ξ9. Para esa "pirueta" podía haberse seguido la vía traumática e incomprensiva con el principiante de adelantar sin más aquel parágrafo o el camino mucho más correcto y elegante de construir un "pasillo" o un "puente" entre ξ9 y ξ30. Se ha puesto en práctica esta segunda posibilidad, dejándola en manos de J. Fernández-Prida.

Quiero concluir este comentario elogiando a la Editorial Tecnos, y ello por motivos que bien se podrían resumir citando dos de los últimos títulos que se ha arriesgado a publicar: *Introducción a la teoría de la computabilidad*, que aquí se comenta, y *¿Qué es la lógica matemática?* de J. N. Crossley y otros. Cuando, editorialmente, se sigue la vía fácil de esquivar –sólo en parte, desde luego– la crisis a costa de cercenar las áreas caras del saber –y la lógica, como las buenas cortesanas, es muy cara–, Tecnos continúa fiel a su amante, a esa amante que paseó con orgullo por las aulas y foros de este país en los tiempos ricos (¡tan lejanos ya!) de *Estructura y Función*. Ahora, sabiamente asesorada por Garrido, Tecnos deja seguir disfrutando ocasionalmente de ella en su *Serie de Filosofía y Ensayo*, aunque en el caso de *Introducción a la teoría de la computabilidad* se haya, paradójicamente, comportado como un musulmán regresivo, vistiéndola de negro hasta dejarnos ver sólo sus ojos (¡por Turing- maravillosos que sean!). La publicación de la versión completa de Hermes-1961 es la única reparación posible al agravio así cometido contra la belleza que yo me atrevería a pedir desde estas páginas. (J. S.)

Bibliografía

Los lectores interesados en una mayor profundización de los temas expuestos pueden consultar los trabajos siguientes:

PROGRAMACION DE ORDENADORES

UNDERSTANDING MEDIA: THE EXTENSIONS OF MAN. Marshall McLuhan. McGraw-Hill Book Company, 1964.

THE MYTHICAL MAN-MONTH: ESSAYS ON SOFTWARE ENGINEERING. Frederick P. Brooks, Jr. Addison-Wesley Publishing Company, Inc., 1974.

MATHEMATICS: THE LOSS OF CERTAINTY. Morris Kline. Oxford University Press, 1980.

THE FRACTAL GEOMETRY OF NATURE. Benoit B. Mandelbrot. W. H. Freeman and Company, 1982.

THE ELEMENTS OF FRIENDLY SOFTWARE DESIGN. Paul Heckel. Warner Books, 1984.

ALGORITMOS Y ESTRUCTURAS DE DATOS

THE HUMBLE PROGRAMMER. Edsger W. Dijkstra en *Communications of the ACM*, vol. 15, n.º 10, págs. 859-866; octubre, 1972.

ALGORITHMS + DATA STRUCTURES = PROGRAMS. Niklaus Wirth. Prentice-Hall Book Company, 1976.

LENGUAJES DE PROGRAMACION

HISTORY OF PROGRAMMING LANGUAGES. Dirigido por Richard L. Wexelblat. Academic Press, 1981.

THE IMPACT OF ABSTRACTION CONCERNS ON MODERN PROGRAMMING LANGUAGES. Mary Shaw en *Studies in Ada Style*, por Peter Shaw, Andy Hisgen, Jonathan Rosenberg, Mary Shaw y Mark Sherman. Springer-Verlag, 1981.

PROGRAMMING LANGUAGES: DESING & IMPLEMENTATION. Terrence W. Pratt. Prentice-Hall, Inc., 1984.

SISTEMAS OPERATIVOS

CONCURRENT EUCLID, THE UNIX SYSTEM AND TUNIS. R. C. Holt. Addison-Wesley Publishing Company, Inc., 1983.

OPERATING SYSTEM DESIGN: THE XINU APPROACH. D. Comer. Prentice-Hall, Inc., 1984.

PROGRAMACION Y TRATAMIENTO DE LENGUAJES

UNDERSTANDING NATURAL LANGUAGE. Terry Winograd. Academic Press, 1972.

METAPHORS WE LIVE BY. George Lakoff y Mark Johnson. University of Chicago Press, 1981.

NATURAL LANGUAGE PROCESSSING. Harry Tennant. Petrocelli Books, Inc., 1981.

THE MENTAL REPRESENTATION OF GRAMMATICAL RELATIONS. Dirigido por Joan Bresnan. The MIT Press, 1982.

LANGUAGE AS A COGNITIVE PROCESS. Terry Winograd. Addison-Wesley Publishing Company, Inc., 1983.

TEX BOOK. Donald E. Knuth, Addison-Wesley Publishing Company, Inc., 1983.

PROGRAMACION DE REPRESENTACIONES GRAFICAS

THE COMPUTER IMAGE: APPLICATIONS OF COMPUTER GRAPHICS. D. P. Greenberg y A. Marcus, Addison-Wesley Publishing Company, Inc., 1982.

FUNDAMENTALS OF INTERACTIVE COMPUTER GRAPHICS. James D. Foley y Andries van Dam. Addison-Wesley Publishing Company, Inc., 1982.

COMPUTER IMAGES: STATE OF THE ART. Joseph Deken. Stewart, Tabori & Chang Publishers, Inc., 1983.

PROCEEDINGS OF THE SIGGRAPH '84 CONFERENCE, JULY 23-27, 1984, MINNEAPOLIS, MINNESOTA. Dirigido por Hank Christiansen en *Computer Graphics*, vol. 18, número 3; julio, 1984.

PROGRAMACION DEL TRATAMIENTO DE LA INFORMACION

AS WE MAY THINK. Vannevar Bush en *The Atlantic Monthly*, vol. 176, n.º 1, págs. 101-108; julio, 1945.

THE ART OF COMPUTER PROGRAMMING, VOL. 3: SORTING AND SEARCHING. Donald E. Knuth. Addison-Wesley Publishing Company, Inc., 1973.

TOWARD PAPERLESS INFORMATION SYSTEMS. F. W. Lancaster. Academic Press, 1978.

PROGRAMACION DEL CONTROL DE PROCESOS

MODERN CONTROL ENGINEERING. Ratsuhiko Ogata. Prentice-Hall Book Company, 1970.

MINI- AND MICROCOMPUTER CONTROL IN INDUSTRIAL PROCESSES: HANDBOOK OF SYSTEMS AND APPLICATION STRATEGIES. Dirigido por M. Robert Skrokov. Van Nostrand Reinhold Company, 1980.

SOFTWARE FOR INDUSTRIAL PROCESS CONTROL. *Computer*, vol. 17, n.º 2; febrero, 1984.

PROGRAMACION EN CIENCIAS Y MATEMATICAS

SMP REFERENCE MANUAL. Stephen Wolfram. Inference Corporation; Los Angeles, 1983.

DOING PHYSICS WITH COMPUTERS. *Physics Today*, vol. 36, n.º 5; 1983.

CELLULAR AUTOMATA: PROCEEDINGS OF AN INTERDISCIPLINARY WORKSHOP, LOS ALAMOS, NEW MEXICO. Dirigido por Doyne Farmer, Tommaso Toffoli y Stephen Wolfram. North-Holland Physics Publishing, 1984.

PROGRAMAS DE SISTEMAS INTELIGENTES

ARTIFICIAL INTELLIGENCE. Patrick Henry Winston. The MIT Press, 1982.

KNOWLEDGE-BASED SYSTEMS IN ARTIFICIAL INTELLIGENCE. Randall Davis y Douglas Lenat. McGraw-Hill Book Company, 1982.

INTELIGENCIA ARTIFICIAL. David L. Waltz en *Investigación y Ciencia*, n.º 75, págs. 48-61; diciembre, 1982.

ARTIFICIAL INTELLIGENCE. Elaine Rich. McGraw-Hill Book Company, 1983.

JUEGOS DE ORDENADOR

PRINCIPLES OF NEURODYNAMICS: PERCEPTRONS AND THE THEORY OF BRAIN MECHANISMS. Spartan Books, 1962.

PERCEPTRONS: AN INTRODUCTION TO COMPUTATIONAL GEOMETRY. Marvin Minsky y Seymour Papert. The MIT Press, 1969.

TALLER Y LABORATORIO

KINEMATICS OF AN ULTRAELASTIC ROUGH BALL. Richard L. Garwin en *American Journal of Physics*, vol. 37, págs. 88-92; 1969.

THE DYNAMICS OF SPORTS. David F. Griffing. Mohican Publishing Company, Loudonville, Ohio; 1982.